AN77900

PSoC® 3 と PSoC 5LP の低電力モードおよび電力低減技術

著者: Greg Reynolds

関連部品ファミリ: すべての PSoC 3 および PSoC 5LP 部品

関連サンプル コード: あり

ソフトウェア バージョン: PSoC Creator™ 4.1 SP1 以降

関連アプリケーション ノートの完全なリストについては、こちらをクリックしてください。

AN77900 は、PSoC 3 および PSoC 5LP の低電力モードおよびそれらの特長について紹介します。主なトピックは、PSoC 電力モード、電力管理の API とレジスタ、省電力技術、および他の低電力モードの考慮事項についてです。関連する PSoC Creator プロジェクトはこれらの原則を示します。

目次

1 はじめに 2
1.1 電力モードとそれらの間の遷移 2
1.2 ウェイクアップ ソース 3
2 アクティブ モードでの電力削減 4
2.1 未使用コンポーネントの電源切断 4
2.2 低速サンプル レートでの ADC の使用 5
2.3 PSoC による電流パスの開閉 5
2.4 クロック速度の動的な変更 6
2.5 DMA によるデータ移動 9
2.6 代替アクティブ モードの使用 9
2.7 電力モード コンフィギュレーション レジスタの使用 9
3 電力モードのその他の考慮事項 10
3.1 高速クロックで電力低減は可能 10
3.2 32kHz 水晶振動子の低電力モード 11
3.3 PSoC スリープ モードでの低電圧割り込み 12
3.4 PSoC スリープ モードでの SegLCD 12
3.5 PSoC 3 スリープ モードでのブースト コンバーター 12
3.6 PSoC 5LP スリープ モードでのブースト コンバーター 13
3.7 高速 IMO の起動 13
3.8 PSoC スリープ モードでのウォッチドッグ 14
3.9 PSoC 低電力モードでの GPIO 14
3.10 PSoC 低電力モードでの SIO 15
3.11 PSoC 低電力モードでのデジタル ブロック 15
3.12 PSoC 低電力モードでの VREF ソース 16
3.13 スリープとハイバネート レギュレータ 17
3.14 プログラミング期間中の消費電力 17
3.15 デバッグ インターフェースは動作しているか？ 18
4 消費電力の概算 18
5 まとめ 18
6 関連アプリケーション ノート 19
付録 A. 低消費電力のサンプル プロジェクト 20
A.1 2 つの低消費電力デモ プロジェクト 20
A.2 電圧アラーム – 最適化なし 21
A.3 電圧アラーム – 最適化あり 22
付録 B. DVK における電力測定 23
B.1 CY8CKIT-001 の変更 23
B.2 CY8CKIT-030 および CY8CKIT-050 の変更 24
付録 C. 電力管理の API およびレジスタ 26
C.1 CyPmSaveClocks() 26
C.2 CyPmRestoreClocks() 26
C.3 CyPmAltAct() 26
C.4 CyPmSleep() 26
C.5 CyPmHibernate() または CyPmHibernateEx() 26
C.6 コンポーネントの低消費電力 API 27
C.7 直接レジスタ書き込み 27
C.8 電力管理 API のフロー チャート 29
C.9 電力管理 API レジスタ リファレンス 30

1 はじめに

PSoC 3 および PSoC 5LP の低電力モードで、特に他の省電力機能や技術と共に適用された場合は、製品の機能を制限することなく、全体的な電流を低減できます。

本アプリケーション ノートは、PSoC 低電力モードの基本を説明し、アクティブ モードでの省電力方法についての情報を提供し、さらに他の低電力に関する考慮事項を記述します。読者が PSoC 3 と PSoC 5LP デバイスのアーキテクチャおよび PSoC Creator に精通していることを前提としています。

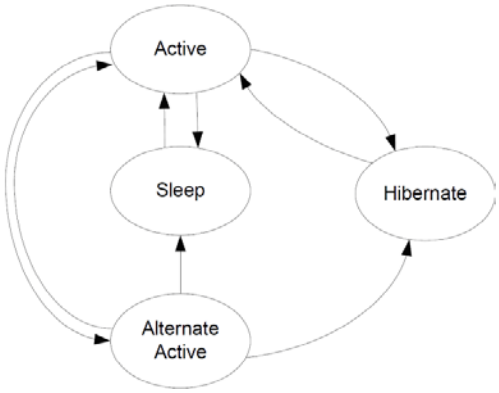
1.1 電力モードとそれらの間の遷移

PSoC 3 および PSoC 5LP デバイスは、アクティブ、代替アクティブ (AltAct)、スリープ、およびハイバネートの 4 種類のモードで機能します。

- アクティブ モードはデバイスの主要な動作モードであり、起動時のデフォルト モードです。一般的に、アクティブ モードは最も多くの電力を消費します。
- AltAct モードはアクティブ モードと似ており、アクティブ モードの代替となる電力コンフィギュレーションです。通常 (アクティブ) モードに復帰する時、デバイスはそれぞれのアナログ／デジタル ブロックを個別にオフにしてから個別にオンにする必要なく、アクティブなアナログ／デジタル ブロックのサブセットに迅速に切り替えられます。初期設定では、CPU は無効です。
- スリープ モードでは、平均電流消費量を減少させる (PSoC 3 の場合は約 1µA 、PSoC 5LP の場合は約 2µA) ために、殆どのサブシステムを無効にします。PSoC 3 と PSoC 5LP の最大ウェイクアップ時間はそれぞれ 15µs と 25µs です。
- ハイバネート モードは、絶対必要な最小限のリソース以外はすべてを無効にし、電力を最大限に節約します (PSoC 3 の場合は約 200nA、PSoC 5LP の場合は約 300nA)。PSoC 3 と PSoC 5LP の最大ウェイクアップ時間はそれぞれ 100µs と 125µs です。

アクティブ モードと AltAct モードは、他のどのモードにも遷移することができます。図 1 に示すように、スリープ モードとハイバネート モードはウェイクアップ時に常にアクティブ モードに復帰します。

図 1. PSoC 3 および PSoC 5LP における電力モード間の遷移



ある電力モードから別のモードへ遷移することは、PSoC 内のすべてのサブシステムの機能に影響を与えます。PSoC Creator が提供する API により、これらの電力モード間の遷移プロセスを単純化し、管理することができます。

1.1.1 アクティブ モード

任意の有効なウェイクアップまたはリセット イベントは、PSoC デバイスをアクティブ モードに戻し、CPU を有効にします。アクティブ モードへの復帰は常に自動的に行われるため、この遷移用には API 関数がありません。

アクティブ モードを終了する通常の方法は、低電力モードの API 関数を呼び出すことです。これらの関数は、PSoC デバイスが低電力モードに入るための準備をし、デバイス全体の電力モード設定を制御するレジスタを更新します。詳細については、「電力管理の API およびレジスタ」を参照してください。アクティブ モードを終了するには必ずしも API 関数を呼び出す必要はありませんが、そうすることを強く推奨します。

1.1.2 AltAct モード

AltAct モードに入る一般的な方法は、API 関数 CyPmAltAct() を呼び出すことです。いずれかの割り込みが保留中状態になると、PSoC デバイスは即座にアクティブ モードに復帰します。

マスクされないウェイクアップ ソースが割り込みを発生させると、PSoC は自動的にアクティブ モードに戻ります。利用可能なウェイクアップ ソースについては、表 1 を参照してください。CPU が AltAct で無効にされていない場合、スリープ モードまたはハイバネート モードに移行することで AltAct モードを終了することもできます。AltAct モードで CPU を有効にする方法については、PSoC 3 レジスタ テクニカル リファレンス マニュアル (TRM) の PM_STBY_CFG0 レジスタを参照してください。

1.1.3 スリープ モード

スリープ モードに入る一般的な方法は、API 関数 CyPmSleep() を呼び出すことです。いずれかの割り込みが保留中状態になると、PSoC デバイスは即座にアクティブ モードに復帰します。

スリープ モード中は CPU と、ほとんどのサブシステムが停止されているため、リセットまたはウェイクアップ イベント以外にスリープ モードを終了する方法はありません。利用可能なウェイクアップ ソースについては、3 ページの表 1 を参照してください。

1.1.4 ハイバネート モード

ハイバネート モードに入る一般的な方法は、API 関数 CyPmHibernate() を呼び出すことです。いずれかのポート割り込み制御ユニット (PICU) の割り込みが保留中状態になると、PSoC デバイスは即座にアクティブ モードに復帰します。

有効な PICU 割り込みまたはハードウェア リセット以外にハイバネート モードを終了する方法はありません。

1.2 ウェイクアップ ソース

ウェイクアップ ソースは、周期、非同期、およびリセットの 3 種類に分類されます。

- 周期的ウェイクアップ ソースには、セントラル タイム ホイール (CTW)、1 パルス毎秒 (OPPS) および LCD タイマーがあります。リアルタイム クロック (RTC) とスリープ タイマー コンポーネントがこれらのタイマーを使用します。
- 非同期ウェイクアップ ソースには、ブースト コンバーター、コンパレータ、I^2C、LVI および PICU があります。
- リセット ウェイクアップ ソースには、外部リセット (XRES) ピンとウォッチドッグ タイマー (WDT) があります。

低電力モードは、これらのウェイクアップ ソースの一部または全部に対応します。表 1 に、各電力モードに対する利用可能なウェイクアップ ソース一覧を表示します。

1.2.1 複数のウェイクアップ ソース

PSoC アプリケーションは複数のウェイクアップ ソースを利用できます。例えば、バッテリ状態確認時 (OPPS)、ボタン押下時 (PICU)、外部温度が高すぎる場合 (コンパレータ) 等に、PSoC デバイスは周期的にウェイクアップする必要があります。

複数のウェイクアップ ソースを設定するには、複数のパラメーターの論理和が取られた電力モード API 関数を呼び出します。ウェイクアップ ソースを判定するために、ウェイクアップ後に割り込みステータス レジスタを読み出さなければなりません。詳細については、「電力管理の API およびレジスタ」を参照してください。

表 1. 低電力モードおよびウェイクアップ ソース

ウェイクアップ ソース	PSoC 3			PSoC 5LP			備考
	Alt Act	スリープ	ハイバネート	Alt AIct	スリープ	ハイバネート	
割り込み	✓			✓			割り込みコンポーネントを使用する必要がある
CTW	✓	✓					スリープ時間は設定可能
スリープ タイマー	✓	✓		✓	✓		スリープ タイマーは CTW を使用
OPPS	✓	✓					32kHz の水晶が必要
RTC	✓	✓		✓	✓		RTC は OPPS を使用

ウェイクアップ ソース	PSoC 3			PSoC 5LP			備考
	Alt Act	スリープ	ハイバネート	Alt Alct	スリープ	ハイバネート	
FTW	✓			✓			
PICU	✓	✓	✓	✓	✓	✓	任意のマスクされないピン割り込み
コンパレータ	✓	✓	✓	✓	✓	✓	
I²C アドレス一致	✓	✓		✓	✓		固定ブロック スレーブ アドレス一致
セグメント LCD リフレッシュ	✓	✓[1]		✓	✓[1]		周期は設定に依存
ブースト コンバーター	✓	✓[2]	[3]	✓	✓	[3]	
WDT	✓	✓		✓	✓		WDT は給電されないと、リセットを発行
LVI	✓	✓		✓	✓		
XRES	✓	✓	✓	✓	✓	✓	デバイスはウェイクアップし、リセット

1. LCD 低電力機能は、CyPmSaveClocks() 関数と共に使用できません。
2. PSoC 3 のスリープ モードでは、ブースト コンバーターをアクティブ モードまたはスタンバイ モードで使用できます。スタンバイ モードが推奨されます。
3. ハイバネート モードは、ブーストを必要とするアプリケーションでは推奨されません。その代わりに、スリープ モードを使用してください。

2 アクティブ モードでの電力削減

ユーザーのアプリケーションは、低電力モードを使用できないか、あるいはアクティブ モードで殆どの時間を費やす場合があります。低電力モードに移行しなくてもアクティブ モードで平均消費電力を低減できます。

2.1 未使用コンポーネントの電源切断

アクティブ モードで消費電力を低減する最も簡単な方法の 1 つは、未使用のコンポーネントの電源を切断することです。

アクティブ モードで無効にすることが可能なコンポーネントは、その API 内に Stop 関数を持っています。この関数を呼び出すと、コンポーネントは直ちにすべての動作を停止し、最小電力状態に入ります。コンポーネントは現時点でタスクを実行していることがあるので、停止する前にその状態を確認してください。

```
/* <Check task status.> */

/* Stop the component. */
MyComponent_Stop();
```

停止されたコンポーネントは、Start 関数を呼び出すことで再度開始できます。

```
/* Start the component. */
MyComponent_Start();
```

電源を切る前にコンフィギュレーション データを保持する必要があるコンポーネントは、その API 内に Sleep 関数を持っています。Sleep 関数はすべての必要なコンポーネント設定を保存してから、Stop 関数を呼び出します。わずかですが Sleep 関数が Stop 関数を呼び出すこと以外は何もしないケースもあります。コード実行時間が重要な場合、その代わりに Stop 関数を呼び出すことができるかを確認するために生成ソース コードを見てください。

```
/* <Check task status.> */

/* Sleep the component. */
MyComponent_Sleep();

/* <Do something else here.> */

/* Wake the component. */
MyComponent_Wakeup();
```

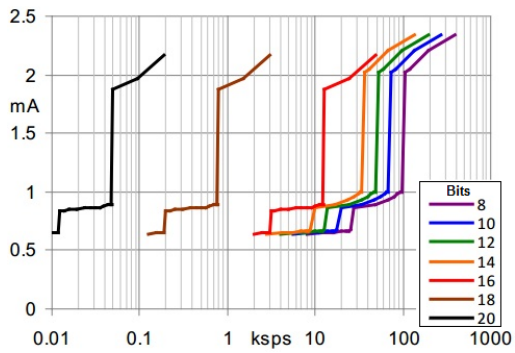
スリープ モードに入ったコンポーネントをウェイクアップするには Wakeup 関数を呼び出すことが必要です。これは、コンポーネントをそのスリープ以前の状態に復元します。Start 関数でコンポーネントは再度動作しますが、デフォルトの状態に再初期化されます。

Sleep 関数も Stop 関数も同じ消費電力の削減量を実現します。違う点は、コンポーネントが正確に終わった所から再開する必要があるかどうかです。本アプリケーション ノートに関連するサンプル プロジェクトは、Stop／Start 関数および Sleep／Wakeup 関数の使用方法を示します。

2.2 低速サンプル レートでの ADC の使用

PSoC での ADC は、幾つかの異なる電力プロファイルを持っています。プロファイルは、分解能とサンプル レートに応じて自動的に有効になります。例えば、図 2 に示すように、電力モードを切り替えると、PSoC 3 での DelSig ADC の消費電力は大幅に変化します。

図 2. PSoC 3 DelSig IDD 対サンプル レート (バッファ付き)



サンプル レートをわずかに低減するだけで電力を著しく節約できます。この場合、ADC が 16 ビット分解能に設定されている場合、10ksps と 12ksps 間の差異はほぼ 1mA です。

アクティブ モードにおける消費電力が重要な場合、コンポーネント データシートで、より遅いサンプル レートが大幅な電力節約を実現できるかを確認することをお勧めします。

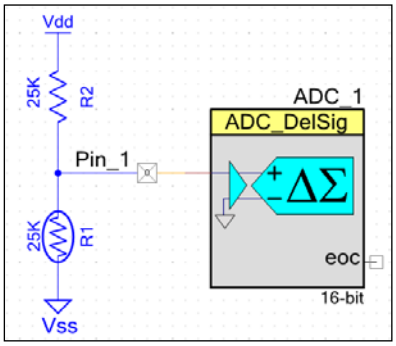
適切なサンプル レートと分解能の選択に関する詳細は、デルタ シグマ ADC コンポーネント データシート、PSoC 3 または PSoC 5LP アーキテクチャ TRM を参照してください。

2.3 PSoC による電流パスの開閉

プリント基板は電力を消費する他のコンポーネントを備えている場合があります。この場合、コンポーネントに流れる電流を制御するために PSoC デバイスを使用できます。データシートに記載されているピンのソースとシンク電流の最大値を超えてはいけないことに注意してください。

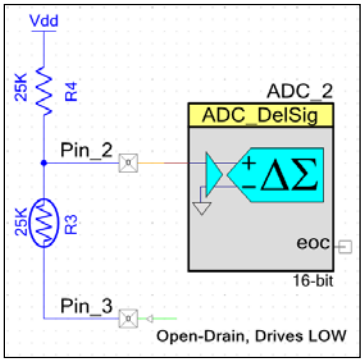
図 3 に示すサーミスタのアプリケーションは、その良い事例です。この場合、PSoC デバイスは、サーミスタ抵抗値の変化に伴って変化するアナログ ピンの電圧で温度を測定します。

図 3. 典型的なサーミスタ アプリケーション



ADC は使用しない時にオフにできますが、抵抗器とサーミスタに流れる電流が停止されないため、外部コンポーネントは依然として電力を消費します。PSoC に対する簡単な解決策は、図 4 に示すように、もう 1 つのピンをグランドへのスイッチとして使用することです。

図 4. GPIO をグランド スイッチとして使用



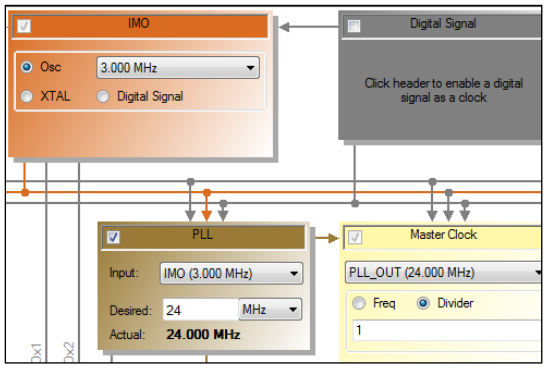
この構成では、Pin_3 に「1」を書き込むことにより、電流の流れを止めることができます。「0」を書き込むと再び電流が流れ始めます。この省電力機能にかかる追加コストは、1 個のピンと数行のコードだけです。

2.4 クロック速度の動的な変更

PSoC 3 および PSoC 5LP は、実行中にクロック速度を変更できます。これにより、ほとんどの時間でクロックを遅く設定し、複雑な動作の実行が必要となる時にクロック速度を上げられます。

図 5 に示すように、新規プロジェクトのクロックの初期設定では、3MHz の内部主発振器 (IMO) が 24MHz の PLL に供給しています。

図 5. 新規プロジェクトの IMO と PLL の初期設定



PLL を無効にし、IMO を 3MHz に設定することで電流消費量を低減できます。これは、あまり中身の無いプロジェクトでは良いですが、目的とするアプリケーションではより速いクロックが必要かもしれません。

例えば、図 6 に示している通りに、1 秒あたりのサンプル数が 10k のマルチ サンプル モードに設定された 16 ビットの ADC は、マスター クロックが最低 12MHz で動作することを必要とします。そうするために、IMO を 12MHz で実行するか、あるいは PLL を最小限 24MHz 以上で実行する必要があります。

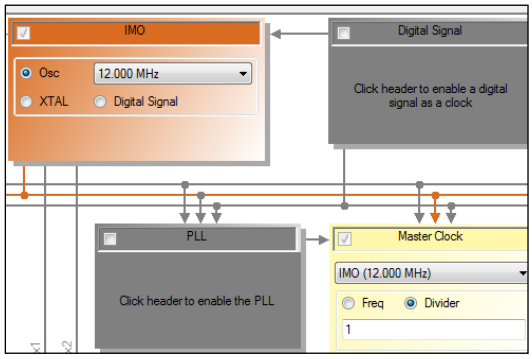
図 6. 警告 – ADC 設定でのクロックが遅すぎる

Name	Desired Frequency	Nominal Frequency	Source Clock
USB_CLK	48.000 MHz	? MHz	IMOx2
Digital_Signal	? MHz	? MHz	
XTAL_32KHZ	32.768 kHz	? MHz	
XTAL	24.000 MHz	? MHz	
PLL_OUT	24.000 MHz	? MHz	IMO
ILO	? MHz	1.000 kHz	
BUS_CLK (CPU)	? MHz	3.000 MHz	MASTER_CLK
MASTER_CLK	? MHz	3.000 MHz	IMO
IMO	3.000 MHz	3.000 MHz	
Clock_1	1.000 kHz	1.000 kHz	Auto: IMO
ADC_DelSig_1_theACLK ⚠	2.590 MHz	3.000 MHz	Auto: MASTER_CLK
ADC_DelSig_1_Ext_CP_Clk ⚠	10.360 MHz	3.000 MHz	Auto: MASTER_CLK

ADC がサンプリング中でない時、プロジェクトは 3MHz に設定された IMO で実行できます。つまり、ユーザーは単に PLL を無効にし、IMO を 3MHz と 12MHz を切り替えられます。

PSoC Creator では、プロジェクト内のすべてのコンポーネントに対応するようにクロックを設定する必要があります。図 7 に示すように、この例ではデフォルト コンフィギュレーションを 12MHz で実行するように設定しなければなりません。

図 7. 動的な変更用のクロック設定



デザインで設定を完了したら、ファームウェアを書き込んで、クロック速度を変更できます。この例では、PSoC を適切に設定するために 3 つの API 関数を使用します。

- CyIMO_SetFreq() – この関数は IMO クロックの周波数を設定します。
- CyFlash_SetWaitCycles() – この関数は、正常のフラッシュ読み書き動作に必要となるウェイトサイクル数を計算し、セットします。
- CyDelayFreq() – この関数は、CyDelay 動作時間を正確に設定するために必要となるサイクル数を計算し、設定します。

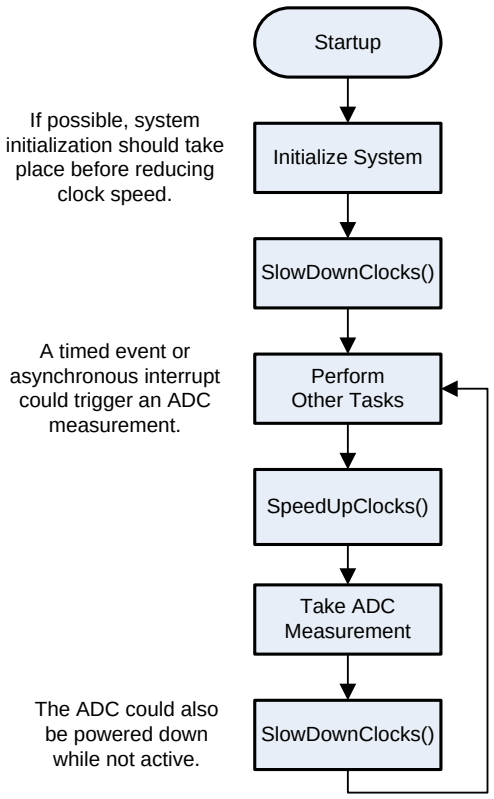
他のクロックは自動的に変更されません。それらの設定を調整するには、コードを追加する必要があります。設定調整に使用する API とレジスタの詳細については、コンポーネント データシートおよびシステム リファレンス ガイドを参照してください。

IMO の速度を変更するために 2 つの関数を作成します。

```
void SlowDownClocks(void)
{
    /* Set IMO frequency to 3MHz. */
    CyIMO_SetFreq(CY_IMO_FREQ_3MHZ);
    /* Set Flash wait to 3MHz. */
    CyFlash_SetWaitCycles(3);
    /* Set CyDelay frequency to 3MHz. */
    CyDelayFreq(3000000);
    /* Change any other active clocks. */
    OtherClock_SetDivider(0); /* 3MHz/1 */
}
void SpeedUpClocks(void)
{
    /* Set IMO frequency to 12MHz. */
    CyIMO_SetFreq(CY_IMO_FREQ_12MHZ);
    /* Set Flash wait to 12MHz. */
    CyFlash_SetWaitCycles(12);
    /* Set CyDelay frequency to 12MHz. */
    CyDelayFreq(12000000);
    /* Change any other active clocks. */
    OtherClock_SetDivider(3);/* 12MHz/4 */
}
```

この例では、システムを初期化した後、直ちにクロックを 3MHz に遅くすることができます。図 8 に示すように、ADC が現にサンプリングしている期間でのみ 12MHz に増加します。

図 8. クロック変更例のフローチャート



低電力モードを使用することなく、他の省電力技術と合わせてクロック速度を調整することで平均電流消費量を低減できます。

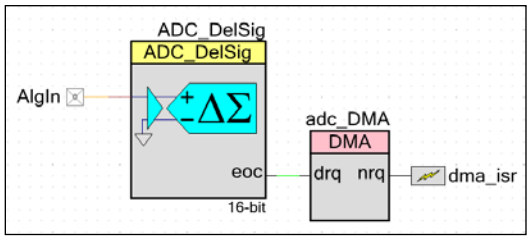
2.5 DMA によるデータ移動

いつでも CPU のタスクを軽減し、それを停止させるか、または他のタスクを並行してやらせることにより、電力を節約できます。PSoC 3 および PSoC 5LP は、CPU 介入なしでアクティブ モードまたは AltAct モードでデータを転送するために使用される DMA エンジンを搭載しています。

図 9 に示す例では、ADC は変換が完了した時に DMA 転送をトリガーします。DMA エンジンは (CPU を使用せずに) ADC の結果を別の位置に移動してから、転送が完了したことを示す割り込みをトリガーします。

DMA の使用は、本書に記述するには範囲が広すぎる複雑なトピックです。サンプル プロジェクトおよび「AN52705 - PSoC® 3 and PSoC 5LP - Getting Started with DMA」といったアプリケーション ノートを含む詳細情報については、www.cypress.com を参照してください。

図 9. ADC 変換完了による DMA トリガー



2.6 代替アクティブ モードの使用

代替アクティブ モードはアクティブ モードとほとんどの機能が似ています。 また、サブシステムを有効または無効にする追加のサブシステム テンプレート ビット セットを持っています。このモードは、アクティブ モードと、CPU を含むデバイスのサブセットが動作している代替低電力モードとの間の迅速な遷移に利用されます。

例えば、テンプレート ビットに書き込むことで CPU を無効にし、代替アクティブ モードで動作するように特定のペリフェラルを有効にします。代替アクティブ モードで割り込みが生成されると、デバイスは自動的にアクティブ モードに移行し、アクティブ モードでファームウェアの実行を開始します。この方法は、個々のブロックの電源をオン／オフにする方法よりはるかに効率的で迅速です。

2.7 電力モード コンフィギュレーション レジスタの使用

アクティブ モードと AltAct モードで、電力モード コンフィギュレーション レジスタを使って、ほとんどのサブシステムへの電源を制御できます。14 の PM_ACT_CFGx レジスタはアクティブ モードで電源とクロックをゲートします。14 の PM_STBY_CFGx レジスタは AltAct モードに適用されます。

PSoC Creator はプロジェクト設定に基づき、電力モード コンフィギュレーション レジスタ用にデフォルト値の一式を生成します。これらの値は *cyfitter_cfg.c* ファイルに格納され、起動時にレジスタにロードされます。実行中に、CY_PM_ACT_CFGx レジスタのいずれかに書き込むことで設定を変更できます。

新しい設定は即座に有効になりますが、リセット後は維持されません。さらに、レジスタの一部のビットは、サブシステムからの割り込みによって自動的に「1」にセットされます。

代わりに、コンポーネント API 内の Stop 関数を呼び出すことを推奨します。これは、常に正しい物理サブシステムにマッピングされているためです。電力モード コンフィギュレーション レジスタのビットマップは、PSoC 3 および PSoC 5LP のレジスタ TRM で説明されています。

3 電力モードのその他の考慮事項

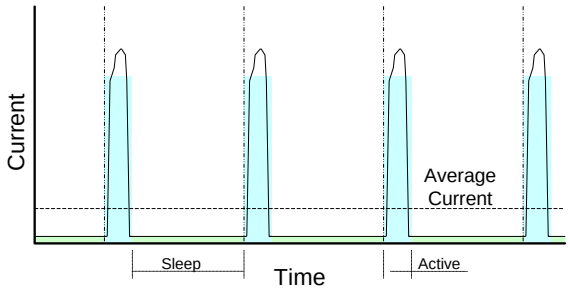
本節は、PSoC 3 および PSoC 5LP の低電力モードの使用に関する種々のヒント、コツ、および推奨事項について説明します。

3.1 高速クロックで電力低減は可能

場合によっては、高速でクロックを実行すると、実際には平均消費電流が少なくなることがあります。例えば、センサーから 1 秒に 1 回読み出し、幾つかの解析と計算を実行してから、結果を別のデバイスに送信する PSoC の設計を検討してみてください。

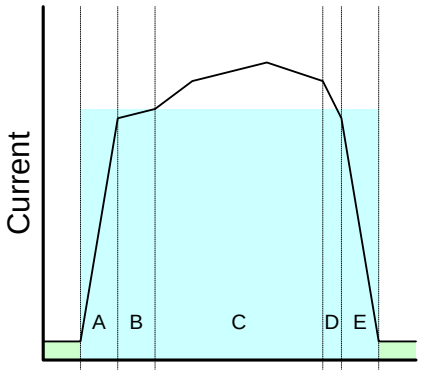
PSoC デバイスがアイドル状態の時、スリープ モードを使用して消費電力を低下させられますが、アクティブ モードで費やす時間が原因で平均電流消費量が大きくなってしまいます。図 10 は、3MHz に設定されたシステム クロックを使ってこの例に該当する電流消費量を説明したものです。

図 10. 3MHz クロックを使った電流分布の例



PSoC デバイスがウェイクアップ状態にある間に実行しているタスクは、システム クロックを速くすることで早く完了することができるかも知れません。これにより、PSoC デバイスはアクティブ モードにある時間が少なくなるため、平均の電流消費量を低減できます。図 11 に、タスクに分けられたアクティブ モードのタイミングを示します。

図 11. 3MHz でアクティブ モードのタスク解析

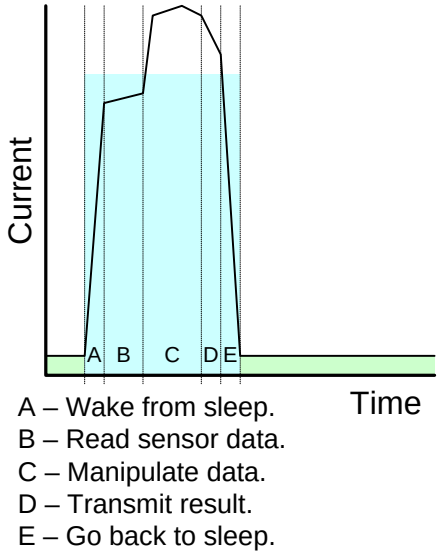


A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

一部のタスクに要する時間は、システム クロックの周波数が増加しても変わりません。センサー読み出しとデータ転送はこのカテゴリに入ります。一方、他のタスクは CPU がより速い周波数で動作すると、処理時間が少なくなります。

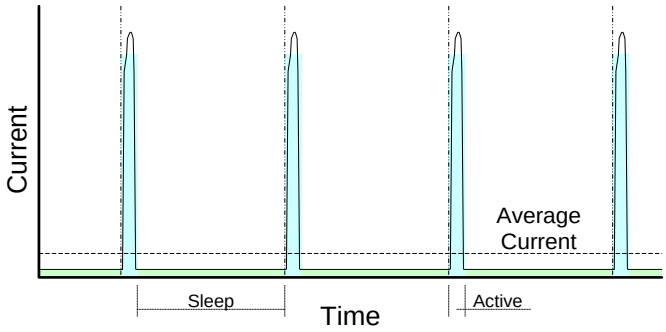
あるポイントで、アクティブ時間が短い利点より、高速でクロックを駆動するために必要となるエネルギーの方が大きくなります。図 12 に示すように、最適速度が 12MHz であると想定します。

図 12. 12MHz でのアクティブ モードのタスク解析



アクティブ モードに費やす時間は低速クロック動作の時と比べると約半分です。図 13 に、クロックが速くなった時、ピーク時の電流消費量が多くなりますが、全体の平均消費量が低くなることを示します。

図 13. 12MHz クロックを使った電流分布の例



本アプリケーション ノートに含まれる他の手法を適用することで、ピーク時のアクティブ電流を低減できる場合があります。

PSoC クロッキングの詳細については、「AN60631 - PSoC 3 and PSoC 5LP Clocking Resources」アプリケーション ノートを参照してください。

3.2 32kHz 水晶振動子の低電力モード

PSoC デバイスがスリープ モードにある時、32kHz 水晶振動子を低電力モードで動作するように設定できます。デフォルトとしては標準電力で動作するように設定されています。この機能を有効にするために、main() 関数の近くにある次の関数を使用します。

```
CyXTAL_32KHZ_SetPowerMode(1);
```

水晶は、アクティブ モードと AltAct モードでは標準電力で動作し続けますが、ハイバネート モード中は無効にされます。省電力モードでは、消費電流は通常電力モードより約 1µA 少ない値です。

システム リファレンス ガイドは、この関数についての詳細情報を提供します。

3.3 PSoC スリープ モードでの低電圧割り込み

低電圧割り込み (LVI) サブシステムは PSoC デバイスをスリープ モードから復帰させるために使用されますが、有効時に約 1µA の電流を消費します。対象とするアプリケーションがアクティブ モード中に LVI を使用するがスリープ モード中には必要としない場合は、それを無効にして電力を節約できます。この機能を無効にするために、次の API 関数を使用します。

```
CyVdLvDigitDisable(); /* Digital LVI */
CyVdLvAnalogDisable(); /* Analog LVI */
```

ウェイクアップ後に再び有効にするために、以下の関数を使用します。

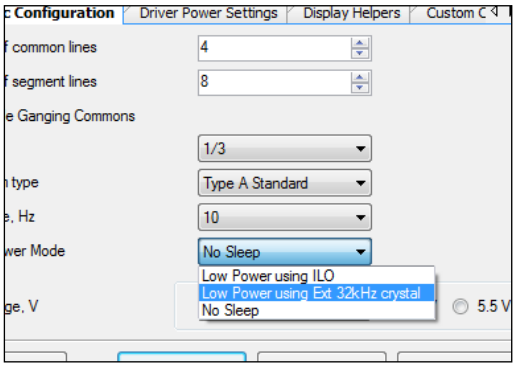
```
CyVdLvDigitEnable(<Reset>,<Threshold>);
CyVdLvAnalogEnable(<Reset>,<Threshold>);
```

初期設定で LVI サブシステムは無効です。システム リファレンス ガイドは、これらの関数についての詳細情報を提供します。

3.4 PSoC スリープ モードでの SegLCD

図 14 に示すように、直接駆動セグメント LCD (SegLCD) コンポーネントは、PSoC デバイスがスリープ モードにある時に LCD セグメントがリフレッシュされることを確保するための低電力のコンフィギュレーションを有しています。

図 14. SegLCD コンフィギュレーション ウィザード



CyPmSaveClocks() 関数は使用できず、SegLCD コンポーネントがスリープ モードで正常に動作するためにはプロジェクトを 12MHz で実行するように設定する必要があります。なぜかと言うと、CyPmSaveClocks() 関数はコンポーネントが使用するデジタル クロックをオフにし、CyPmSleep() 関数はシステム クロックを IMO から 12MHz で動作するように設定するためです。

PSoC Creator は、アクティブ モードとスリープ モードでの動作を説明する 2 件の SegLCD サンプル プロジェクトを提供しています。スリープ モードを使用するシステムに SegLCD コンポーネントを実装する詳細については、それらのサンプル プロジェクトを参照してください。詳細情報は、「AN52927 – PSoC 3 and PSoC 5LP Segment LCD Direct Drive」を参照してください。

3.5 PSoC 3 スリープ モードでのブースト コンバーター

PSoC 3 ブースト コンバーターは PSoC 3 のすべての電力モードで使用できます。2 つの動作モードがあります。

- ブースト アクティブ モードはアクティブ モードまたは AltAct モードで使用されます。このモードでは、ブースト コンバーターは出力電圧を監視し、PSoC の全機能に対応できます。
- ブースト スタンバイ モードは、PSoC デバイスがスリープ モード中に動作するのに十分な電力を供給する低電力状態です。

チップのハイバネート モードをブースト コンバーターに使用することは推奨しません。ブースト コンバーターがチップの残りの部分より多くの電力を消費しているため、スリープ モードよりハイバネート モードの方が少ない電力削減量です。

ブースト コンバーターがブースト スタンバイ モードにある時、PSoC デバイスはアクティブ モードまたは AltAct モードで (コンフィギュレーションによって) 数マイクロ秒しか動作できません。PSoC デバイスをスリープ状態にする直前に、ブースト モードをスタンバイ モードに変更してください。また、ウェイクアップ後にできるだけ早くブースト モードをアクティブ モードに設定する必要があります。

```
/* Set system clocks for low power. */
CyPmSaveClocks();
/* Set boost to Standby mode. */
BoostConv_SetMode(BoostConv_BOOSTMODE_STANDBY);
/* Sleep PSoC 3 until wakeup event. */
CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_BOOSTCONVERTER);
/* Restore boost to Active mode. */
BoostConv_SetMode(BoostConv_BOOSTMODE_ACTIVE);
/* Restore system clocks. */
CyPmRestoreClocks();
```

スリープ モードでのコンバーターの自動更新、または「サンプ」時間を計るために外部の 32kHz 水晶を使用します。サンプは、API 関数がブースト コンバーターをスタンバイ モードにするために使用される場合は、デフォルトで有効にされます。

ブースト サブシステムに関する詳細情報は、PSoC 3 データシートと TRM を参照してください。ブースト コンバーター コンポーネントの詳細は、ブースト コンバーター コンポーネント データシートおよび システム リファレンス ガイドを参照してください。

3.6 PSoC 5LP スリープ モードでのブースト コンバーター

PSoC 5LP ブースト コンバーターは PSoC 5LP のすべての電力モードで使用できます。2 つの動作モードを持っています。

- ブースト アクティブ モードはチップのアクティブ、AltAct、およびスリープ モードで使用されます。このモードでは、ブースト コンバーターは出力電圧を監視し、PSoC の全機能に対応できます。
- ブースト スタンバイは、PSoC デバイスがスリープ モード中に動作するのに十分な電力を供給する低電力状態です。

ブースト サブシステムをハイバネート モードのチップで使用することは推奨しません。ブースト コンバーターがチップの残りの部分より多くの電力を消費しているため、スリープ モードよりハイバネート モードの方が少ない電力削減量です。

ブースト コンバーターがスタンバイ モードにあり、PSoC デバイスに給電している時、PSoC デバイスはアクティブ モードまたは AltAct モードで (コンフィギュレーションによって) 数マイクロ秒しか動作できません。PSoC デバイスをスリープ状態にする直前に、ブースト モードをスタンバイ モードに変更してください。

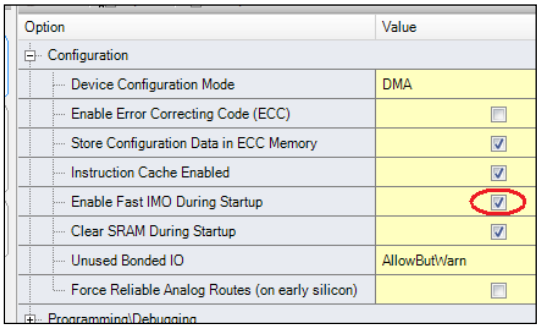
```
/* Set system clocks for low power. */
CyPmSaveClocks();
/* Set boost to Standby mode. */
BoostConv_SetMode(BoostConv_BOOSTMODE_SLEEP);
/* Sleep until boost refresh is needed. */
CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_CTW);
/* Restore boost to Active mode. */
BoostConv_SetMode(BoostConv_BOOSTMODE_ACTIVE);
/* Restore system clocks. */
CyPmRestoreClocks();
```

ブースト サブシステムに関する詳細情報は、PSoC 5LP データシートと TRM を参照してください。ブースト コンバーター コンポーネントの詳細は、ブースト コンバーター コンポーネント データシートおよび システム リファレンス ガイドを参照してください。

3.7 高速 IMO の起動

PSoC 3 および PSoC 5LP は、起動を高速化するために IMO を 48MHz で開始する高速 IMO (FIMO) 機能を備えています。即ち、起動時は通常動作時より電流消費量が多いことを意味します。この機能は、図 15 に示すように、PSoC Creator プロジェクトの Design-Wide Resources ファイルでデフォルトとして有効にされています。

図 15. .cydwr タブの高速 IMO 選択



この機能を無効にすると、起動時の電流消費量を削減できますが、起動時間と初期化時間が遅くなってしまいます。FIMO 機能と起動時間への影響の詳細については、PSoC 3 と PSoC 5LP アーキテクチャ TRM の「Clocking System」節を参照してください。

3.8 PSoC スリープ モードでのウォッチドッグ

ウォッチドッグ タイマーはアクティブ、AltAct、およびスリープ モードで動作します。低電力ウォッチドッグの動作に次の 3 つのオプションがあります。

- 変化なし – ウォッチドッグは指定した間隔で実行し続け、間隔が終了する前にクリアされる必要があります。
- 最大間隔 – ウォッチドッグは実行を継続し、クリアされる必要がありますが、間隔は最大値 (1024 ティック) に設定されます。ウェイクアップ後の最初のクリアで元の間隔に戻ります。
- 無効 – ウォッチドッグは PSoC デバイスが低電力モードにある時に無効にされます。ウェイクアップすると再び有効になり、指定した間隔で実行します。

ウォッチドッグが期待通りに設定されていることを確実にするために API (CyWdtStart) を使用することを推奨します。「最大間隔」オプションは、PM_WDT_CFG レジスタでデフォルトとして設定されています。

ウォッチドッグはハイバネート モードで非アクティブです。ハイバネート モードから復帰するとリセットされますが、ユーザーファームウェアで定義した設定通りに動作しない可能性があります。

ウォッチドッグ タイマーの動作と関連 API の詳細情報は、PSoC 3 と PSoC 5LP のアーキテクチャ TRM およびシステム リファレンス ガイドを参照してください。

3.9 PSoC 低電力モードでの GPIO

GPIO は、PSoC デバイスが低電力モードにある時に駆動し続けることができます。他の外部ロジックを固定レベルに維持する必要がある時には役立ちますが、ピンが電流を不必要に流し出したり、引き込んだりする時には、電力の無駄になります。

設計を分析して、低電力動作時の GPIO 用に最適な状態を決める必要があります。デジタル出力ピンを論理 1 か論理 0 に維持するのが最適であれば、コンポーネントの書き込み関数を使用して設定してください。

```
/* Set My Pin to '0' for low power. */
MyPin_Write(0);
```

別の駆動モードを使用する特別な理由がない限り、すべての未使用の GPIO をアナログ Hi-Z に設定する必要があります。ピン コンポーネントに関連するすべての物理ピンをアナログ Hi-Z に設定できる場合、コンポーネントの SetDriveMode 関数を使用することが可能です。

```
/* Set My Pin to Alg Hi-Z for low power. */
MyPin_SetDriveMode(MyPin_DM_ALG_HIZ);
```

ピン コンポーネントに関連する一部のピンのみを変更する場合は、ポート ピン コンフィギュレーション レジスタに書き込んでください。PSoC デバイスの GPIO ピンごとに、CYREG_PRTx_PCy と名付けられた 1 つのレジスタがあります (x = ポート番号、y = ピン番号)。

PSoC 3 と PSoC 5LP の柔軟性により、望ましくない電流漏れを防止する GPIO 駆動モードを、簡単に管理できます。詳細については、「AN72382– Using PSoC 3 and PSoC 5LP GPIO Pins」を参照してください。

3.10 PSoC 低電力モードでの SIO

PSoC スリープ電流を減少させるために、PSoC デバイスをスリープ モードにする前に、特殊入出力 (SIO) ピンをシングルエンド モードに入れる必要があります。差動モードでの SIO は、100µA の高電流を消費します。SIO をシングルエンド モードにするために、PRT12_SIO_CFG レジスタを使用して特定 SIO ペアのビットを 0 にセットします。PSoC デバイスがスリープモードから復帰した後、それらのビットを以前の値に設定し直す必要があります。表 2 で、SIO コンフィギュレーション レジスタを説明します。

表 2. PRT12_SIO_CFG レジスタ

SIO[7:6]		SIO[5:4]		SIO[3:2]		SIO[1:0]	
ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0

詳細については、「AN60580 – SIO Tips and Tricks in PSoC 3/PSoC 5LP」を参照してください。

また、SIO ピンを出力として使用し、「駆動レベル」を「Vddio」の代わりに「Vref」にセットした場合、電力を最小限に抑えるために低電力モードで Vref モードを無効にする必要があります。これはペアで行い、2 つの隣接ピンから成る一組で Vref 出力モードを有効にします。以下に、ピン ペアのマスクを定義するサンプル コードおよび Vref を有効／無効にする方法を示します。

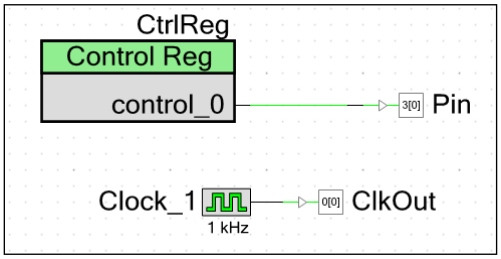
```
#define SIO_VREG_EN_1_0 0x01 // Mask for SIO pins[1:0]
#define SIO_VREG_EN_3_2 0x04 // Mask for SIO pins[3:2]
#define SIO_VREG_EN_5_4 0x10 // Mask for SIO pins[5:4]
#define SIO_VREG_EN_7_6 0x40 // Mask for SIO pins[7:6]

// Code to enable and disable Vref output mode (Instance name is “MySIO”)
MySIO_SIO_CFG |= VREG_EN_3_2; // Set to use Vref reference
MySIO_SIO_CFG &= ~VREG_EN_3_2; // Clear Vref for Low Power mode
```

3.11 PSoC 低電力モードでのデジタル ブロック

幾つかのサブシステムは、スリープ モードとハイバネート モードで常に電源オフです。それらを電源供給されている他のサブシステムに接続すると、好ましくない動作を引き起こす可能性があります。

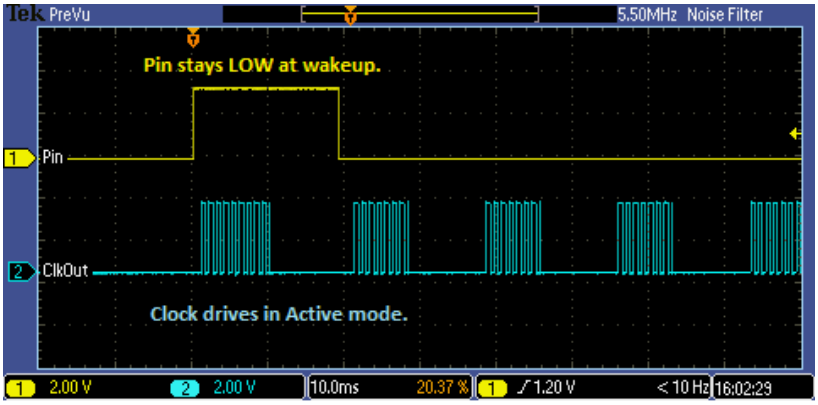
図 16 は、GPIO ピンの出力を設定するために使用される制御レジスタを示します (他のピンとクロックは PSoC デバイスがウェイクアップ状態にあることを示します)。

図 16. ピン出力の設定に用いる制御レジスタ



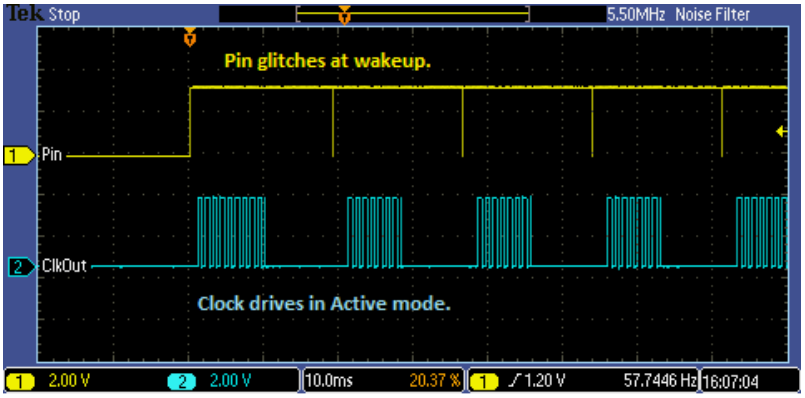
チップが低電力モードにある間、ピンは最後の状態 (HIGH) を駆動し続けますが、制御レジスタを含む物理ブロックは電源が供給されません。PSoC デバイスがウェイクアップし、デジタル ブロックへの電力が回復すると、制御レジスタ ビットは「0」にリセットされます。その後、図 17 に示すように制御レジスタに接続されているピン出力は HIGH から LOW に切り替わります。

図 17. 制御レジスタはスリープ中に常に一定ではない



制御レジスタをウェイクアップ時に「1」にセットするファームウェアを追加したとしても、グリッチが発生します。これは、図 18 に示すように、ファームウェアがその状態を変更する前に、ウェイクアップ時にピンは短時間「0」となるためです。

図 18. 各ウェイクアップ時の制御レジスタのグリッチ



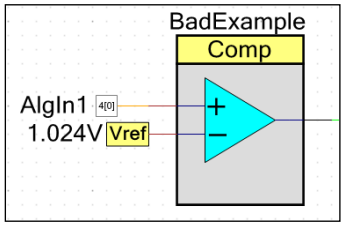
この場合、制御レジスタを使用するより、ピンを直接制御した方が良いです。

3.12 PSoC 低電力モードでの VREF ソース

PSoC では、スリープ モードまたはハイバネート モード中に VDDA、VDDD、および VBAT は接続されたままですが、他の VREF ソースはこの限りではありません。さらに、VDDA/2 リファレンスはアクティブのままですが、電圧分周器は切断されます。スリープ モードまたはハイバネート モード中にリファレンス電圧が必要となる場合は、外部ソースを使用しなければなりません。

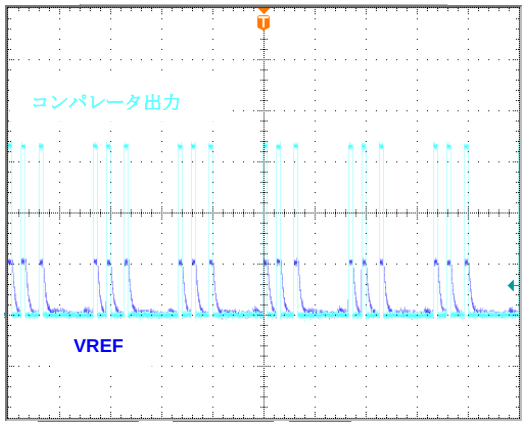
図 19 に、このような場合の例を示します。コンパレータはウェイクアップ ソースに設定され、負端子に接続した VREF コンポーネントを持っています。

図 19. 低消費電力で Vref を使用しない



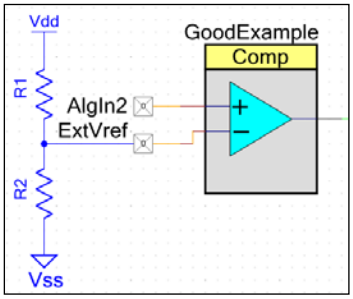
これは、PSoC デバイスがアクティブ モードまたは AltAct モードにある限りは大丈夫ですが、低電力モードになると VREF が切断されます。その結果、負端子が開放された時、図 20 に示すように断続的なウェイクアップとなります。

図 20. コンパレータ入力開放時の断続的なウェイクアップ (電圧スケール = 1.0V)



この問題の解決策は、図 21 に示すように外部リファレンス電圧を供給することです。このリファレンス電圧は PSoC デバイスがスリープ モードにある間にアクティブのままであるため、コンパレータ ウェイクアップ ソースは期待通りに機能します。

図 21. 低消費電力モードで外部 VREF を使用



入力端子ではごく小さな電流しか必要としないため、分周器内で失われる電力を制限するために大きな抵抗を使用できます。

3.13 スリープとハイバネート レギュレータ

PSoC 3 および PSoC 5LP は、スリープ モードとハイバネート モードで論理状態を維持するための 2 個の低電力レギュレータを備えています。

- スリープ レギュレータは、高速ウェイクアップ用に必要な、またスリープモード中にアクティブ状態を維持するサブシステムが必要とする電力を供給しています。
- ハイバネート レギュレータは、ハイバネート モードの期間中には必要不可欠なレジスタ、メモリおよびラッチの論理状態を保持するために十分なだけの電力を供給します。

スリープ レギュレータの出力は VCCD と VCCA ピンで見られますが、内部 PSoC リソース以外に何にも十分な電流を提供しません。スリープ レギュレータを他のどの目的にも使用しないでください。

ハイバネート レギュレータは、VCCD と VCCA ネットに電源を供給しないため、その出力を見ることはできません。

3.14 プログラミング期間中の消費電力

PSoC 3 および PSoC 5LP は、プログラミング期間中に約 14mA の電流を消費します。これは、プログラミングとデバッグロジックがクロックと CPU と共に有効にされるためです。

このレベルの電流を提供できないデザインの場合は、プログラミングやデバッグ中に MiniProg3 などの外部ソースから PSoC デバイスに電源を供給する必要があります。PSoC に電源供給するために MiniProg3 をどのように設定するかの情報については、PSoC Creator のヘルプ ファイルを参照してください。

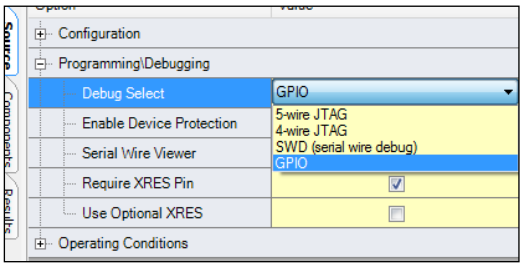
3.15 デバッグ インターフェースは動作しているか？

PSoC 3 および PSoC 5LP はオンチップ デバッグに対応しています。デバッグ モードの期間中に、予想したより高い消費電流を観測することがあります。プログラミングおよびデバッグ インターフェースが低電力モードでアクティブのままなので、これは正常です。

PSoC デバイスがデバッグ モードでない場合であっても、デバッグ ピンが SWD モードあるいは JTAG モードに設定され、MiniProg3 が接続された時、電力測定に偏りが生じる場合があります。

デバッグ インターフェース ピンは工場出荷時にすべてのチップで GPIO モードに設定されていますが、新規 PSoC Creator プロジェクトではデフォルトで SWD モードに設定されます。デバッグ インターフェースを制御するレジスタは、プログラミング時にのみ変更可能です。図 22 に示すように、PSoC Creator プロジェクトの*.cydwr* ファイル内の System タブを使用してピンを GPIO モードに設定します。

図 22. 電力削減のためにデバッグ インターフェースを無効化



ピンが GPIO モードに設定されても、プログラミングおよびデバッグは依然として可能です。デバッグ インターフェースが無効にされた場合は、PSoC デバイス内のデバッグ コントローラーにアクセスするためにリセットを行う必要があります。つまり、ユーザーができない唯一のことは、実行中のプロジェクトにデバッガを接続することです。

いかなるリリースされた製品に対しても、デバッグ インターフェース ピンを GPIO モードに設定することを推奨します。プログラミングおよびデバッグの詳細については、デバイスのデータシートおよび TRM を参照してください。

4 消費電力の概算

デバイス データシートおよびコンポーネント データシートは、特定のプロジェクトに対して消費電力を見積もるために十分な情報を提供しています。このプロセスを簡単にするために、広範な内部コンポーネントについての一般的な電力要件を記載したスプレッドシートが用意されています。PSoC3_5_Power_Estimator.xls スプレッドシートは本アプリケーション ノート AN77900 を提供するサイプレスのウェブページに掲載されています。プロジェクトによって違うため、スプレッドシートが提供する電力計算はあくまでも目安ですが、デザインを完了する前に意味のあるフィードバックを提供するために十分近い数値である必要があります。スプレッドシートに幾つかのタブがあります。データを入力する前に、**Instructions** タブを読むことを忘れないでください。

5 まとめ

適切な電力削減技術は、携帯機器に大きく影響します。デバイス バッテリのサイズが縮小され、その結果、製品のサイズが縮小したり、あるいは特定のバッテリの寿命が延びます。PSoC 3 および PSoC 5LP で利用可能な多くの省電力機能を活用することにより、ユーザー独自のデザインを最適化し、最低限の消費電力を保証することができます。

6 関連アプリケーション ノート

下記のアプリケーション ノートは、本書で完全に説明されていないトピックについての詳細情報を提供します。

- AN86233 – PSoC 4 Low-Power Modes and Power Reduction Techniques
- AN54181 – Getting Started with PSoC 3
- AN77759 – Getting Started with PSoC 5LP
- AN77835 – PSoC 3 to PSoC 5LP Migration Guide
- AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations
- AN54460 – PSoC 3 and PSoC 5LP Interrupts
- AN60616 – PSoC 3 and PSoC 5LP Startup Procedure
- AN60631 – PSoC 3 and PSoC 5LP Clocking Resources
- AN72382 – Using PSoC 3 and PSoC 5LP GPIO Pins
- AN60580 – SIO Tips and Tricks in PSoC 3/PSoC 5LP
- AN52705 – PSoC 3 and PSoC 5LP– Getting Started with DMA
- AN52927 – PSoC 3 and PSoC 5LP – Segment LCD Direct Drive

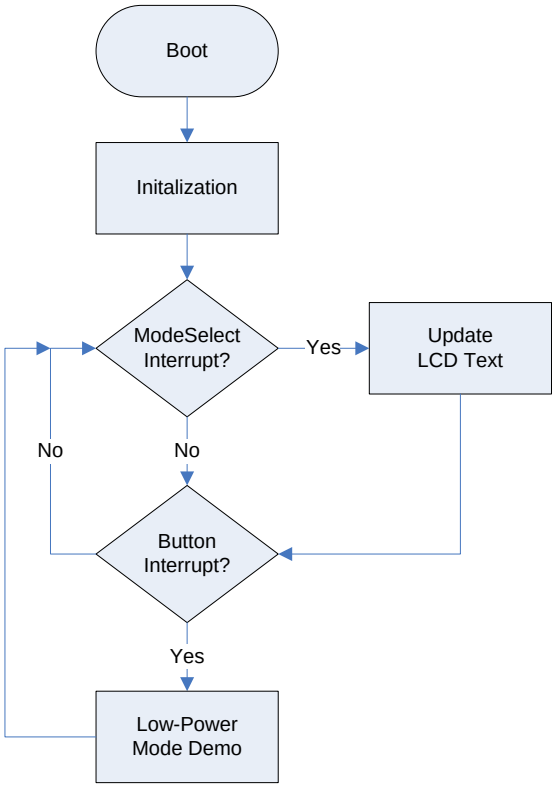
付録 A. 低消費電力のサンプル プロジェクト

PSoC に精通する一番良い方法は、それを実際に使ってみることです。本アプリケーション ノートに関連するサンプル プロジェクトが 4 つあります。それらのプロジェクトはすべて同じ PSoC Creator のワークスペースの一部であり、同じ開発キット (DVK) ハードウェア セットアップで実行できます。

A.1 2 つの低消費電力デモ プロジェクト

これらのプロジェクトは PSoC 3 と PSoC 5LP の電力モード遷移とウェイクアップ ソースについて説明します。ポート 0 のピンを指定された値に設定することで電力モードとウェイクアップ ソースを選択してください。図 23 に示すように、P2[7]の立ち下がりエッジにより、PSoC デバイスがコードを実行して選択したコンフィギュレーションをデモします。

図 23. PSoC 低消費電力デモフロー チャート



本プロジェクトはすべての共通のスリープとハイバネート ウェイクアップ ソースについて説明します。SegLCD は PSoC Creator に独自のサンプル プロジェクトを持っており、スリープ モード時のブースト調整は本アプリケーション ノートの本文で説明されています。全体的にはスリープ モードにおいてとは同じであるため、2 つの AltAct ソースについてのみ提示しています。表 3 は、これらのサンプル プロジェクトによって説明されるモードおよび対応するウェイクアップ ソースの一覧です。

表 3. PSoC デモのウェイクアップ ソース

モード	ウェイクアップ ソース
アクティブ	なし
AltAct	PICU
	RTC
スリープ	PICU
	RTC (OPPS を使用)
	SleepTimer (CTW を使用)
	CTW (PSoC 3 のみ)
	コンパレータ
	I^2C アドレス
	LCD (未実装)
ハイバネート	PICU
カスタム	なし (ユーザー定義)

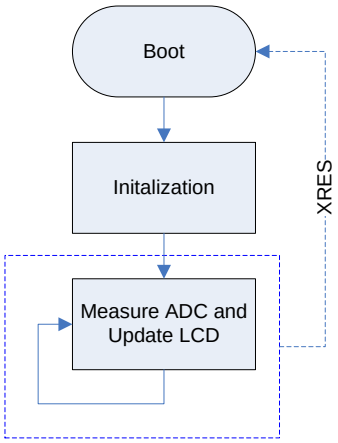
このコードは本アプリケーション ノートで説明した原理と技術を提示しています。サイズや実行スピードの最適化は行っていません。プロジェクトを使用するための説明はプロジェクトの回路図とソース ファイルにあります。

A.2 電圧アラーム – 最適化なし

本プロジェクトは、図 24 に示すように、PSoC 3 デバイスを使用した簡単な電圧測定およびアラームのシステムをデモします。このプロジェクトは低消費電力に最適化されていません。デフォルトのクロックとグローバル電源の設定は変更されず、何の低消費電力モードも使用されていません。

DelSig ADC はアナログ入力の値を読み出し、結果が LCD に表示されます。RTC コンポーネントも動作し、LCD に時間の値が表示されます。入力の電圧が定義されたレベルを超えると、LED がパルスを開始し、「アラーム」が LCD に表示されます。

図 24. アラーム システム – 最適化なし

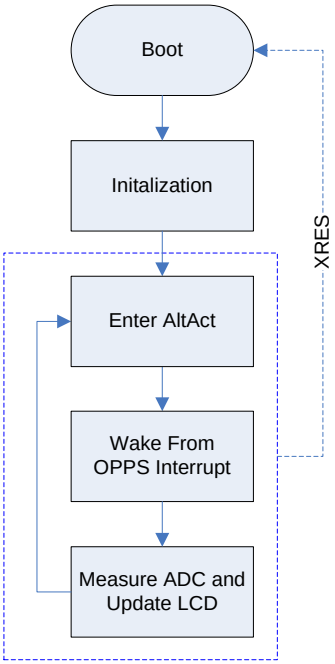


プロジェクトの使用説明はプロジェクトの回路図とソース ファイルに記載されています。

A.3 電圧アラーム – 最適化あり

このプロジェクトは、前の電圧測定およびアラームと同じ基本機能をデモします。図 25 に示すように、消費電力低減のためにいくつかの低電力最適化が適用されています。

図 25. アラーム システム – 最適化あり



プロジェクトの使用説明はプロジェクトの回路図とソース ファイルに記載されています。

付録 B. DVK における電力測定

サイプレス開発キット (DVK) は PSoC デバイスに備えているアナログおよびデジタル機能をデモするように設計されています。PSoC デバイスによってのみ消費される電流を容易に測定するためには最適化されていません。本付録では、正確な PSoC 電力測定のために CY8CKIT-001、CY8CKIT-030 および CY8CKIT-050 基板を変更する方法について説明します。変更を行う前に、基板の回路図に精通することをお勧めします。回路図は、個々のキットのウェブページから入手できます。

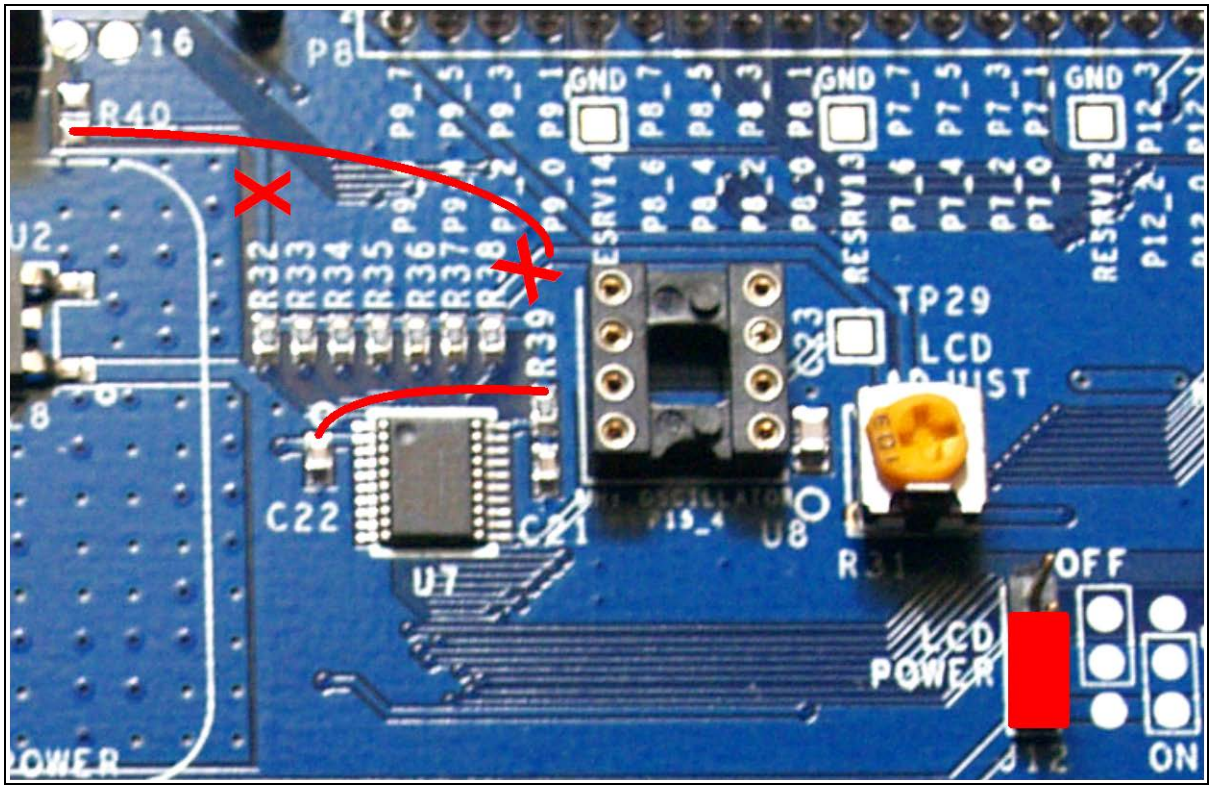
B.1 CY8CKIT-001 の変更

次の手順は、適切な電流測定を確実に行うために CY8CKIT-001 DVK 基板を変更する方法を示します。ここで説明するコンフィギュレーションは、VDDD、VDDA および VDDIO に単一電源レールを使用します。電力はメーターを PSoC の VDDD ピンと直列に配置することで測定されます。

1. J2、J3、J4、J5、J6、J7、J10 および J11 からジャンパを取り外します。
2. J7 のセンターピンに J6 のセンターピンを短絡します。これにより、VDD_DIG と VDD_ANLG ネットを互いに接続します。
3. J6 のセンターピンに J2、J3、J4、J5 のセンターピンを短絡します。これにより、VDD_DIG を VDDIO ネットに接続します。
4. J11 のピン 1 (「VR」テキストの上) を J10 のピン 1 (「RS232」テキストの下) に短絡します。これにより、VDD_ANLG の代わりに VDD ネットから R20 のポテンション メーターに電源供給します。
5. マルチメーターからの LOW 側のリードを J7 のセンターピンに接続します。
6. マルチメーターからの HIGH 側のリードを VDD テスト ポイントに接続します。

   注: 5V で動作している場合、さらなる変更は不要です。5V 以下で動作している場合、LCD レベル シフター回路は PSoC I/O ピンに電流を供給し、スリープまたはハイバネート モード時の消費電流の測定値を歪めることがあります。次の手順では、このようなことが起こらず、PSoC デバイスが 3.3V または 5V で動作している時にいつでもキャラクタ LCD が動作できるようにします。この変更は、プリント基板上の配線の切断を含むことをあらかじめ注意してください。24 ページの図 26 は、次の手順で説明する切断とジャンパ接続を示します。
7. DVK 基板の再上面の「R32」テキストを通る配線を切断します。近くにビアや部品がない R40 に向かって曲がる前にテキストを通ってすぐの所で切断することをお勧めします。
8. 同様に、「R38」テキストと「RESRV14」テキストを通る配線の間の所で切断します。2 回の切断は HIGH 側のシフター信号上のプルアップ抵抗を VCC_LCD ネットから分離します。
9. 「RESRV14」テキストに最も近い側で、切断した配線からソルダー レジストを削り取り、はんだ付けのために銅を露出させます。
10. R40 の下部パッド (「R40」テキストの下) から、前のステップでソルダー レジストが削り取られた領域までジャンパ線をはんだ付けします。これは、前の 2 回の切断により分離された部分がバイパスされ、LCD モジュールの電源が J12 のセンター ピンに接続されます。
11. R39 の上部パッド (「R39」テキストの下) から C22 の下部パッド (U7 の丸いシルクスクリーンの隣) までジャンパ線をはんだ付けします。そうすると、HIGH 側のシフタのプルアップとリファレンス ピンが 3.3V 電源に接続されます。
12. LCD モジュールを有効にするために J12 のジャンパを ON の位置に設定します。

図 26. 3.3V での低消費電力測定のための CY8CKIT-001 LCD レベル シフト バイパス



変更を行った後、PSoC デバイスが消費する全電流を正確に測定できます。KIT-001 基板上の他の部品はすべて VDDD とは独立して給電されるので、電力測定には現れません。CY8CKIT-001 は SW3 を使用することにより、これ以上の変更なしに 5V または 3.3V の動作に設定できます。

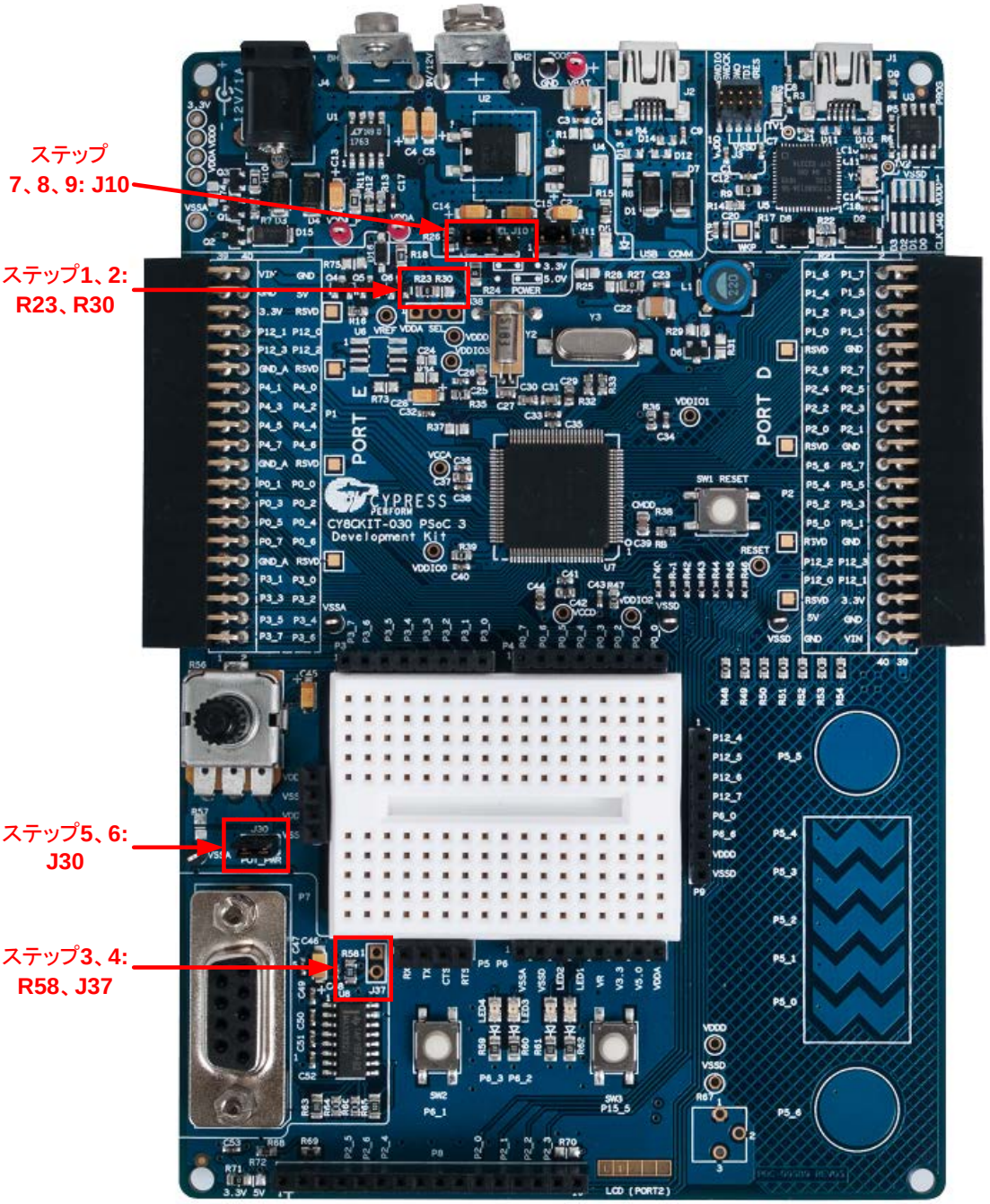
B.2 CY8CKIT-030 および CY8CKIT-050 の変更

次の手順は、適切な電流測定を確実に行うために CY8CKIT-030 および CY8CKIT-050 DVK 基板を変更する方法を示します。使用するハードウェアは 25 ページの図 27 に示します。ここで説明するコンフィギュレーションは、VDDD、VDDA および VDDIO に単一電源レールを使用します。電力はメーターを PSoC の VDDD ピンと直列に配置することで測定されます。

1. R23 を取り外します。そうすると、VDDA_P ネットから VDDA ネットが分離されます。
2. R30 のパッドに 0Ω の抵抗を追加します。そうすると、VDDD が VDDA に接続され、PSoC デバイスの全消費電流を測定することができます。
3. R58 を取り外します。そうすると、VDDA から RS-232 ドライバーU8 が分離されます。デフォルトでは、U8 は VDDA から電源供給されるので、電力測定に現れてきます。
4. RS-232 が必要な場合、VDDA_P テスト ポイントに J37 のピン 2 (「J37」テキストの上) を短絡させます。そうすると、VDDA の代わりに VDDA_P から RS-232 ドライバーが電源供給され、RS-232 ドライバーは電力測定に現れません。
5. J30 からジャンパを取り外します。デフォルトでは、R56 は VDDA から電源供給されるので、電力測定に現れます。
6. ポテンションメーターR56 が必要な場合、VDDA_P テスト ポイントに J30 のピン 2 (「J30」テキストの下) を短絡させます。そうすると、ポテンショメーターは VDDA_P から給電されます。
7. J10 からジャンパを取り外します。
8. マルチメーターの LOW 端子を J10 のセンターピンに接続します。
9. マルチメーターの HIGH 端子を J10 の (5V または 3.3V 動作用) 外側ピンの 1 本に接続します。

変更を行った後、PSoC デバイスが消費する全電流を正確に測定できます。DVK 基板上の他のコンポーネントはすべて VDDD とは独立して給電されるので、電力測定には現れません。J10 と J11 を使用することにより、DVK は追加の変更なしに 5V または 3.3V の動作に設定できます。

図 27. CY8CKIT-030／050 の低消費電力変更



付録 C. 電力管理の API およびレジスタ

サイプレスは、PSoC デバイスの低電力モード間の遷移を処理するための API ルーチンを提供しています。ここで説明する機能は、最も一般的に使用されるものです。すべての PSoC Creator プロジェクトの一部である *CyPm.c* に用意されています。

システム リファレンス ガイドに電力管理 API を詳細に説明する節があり、PSoC 3 および PSoC 5LP の TRM にはレジスタの更なる情報が含まれています。

C.1 CyPmSaveClocks()

この関数は、低電力動作のために PSoC クロックの準備をします。スリープまたはハイバネート モードに入る直前に呼び出しなければなりません。クロックが動作したままであることが想定されるため、通常は AltAct モードに入る前には呼び出しません。

スリープまたはハイバネートモードに入る前にこの関数の呼び出しができない場合、不定の動作が発生する可能性があります。

C.2 CyPmRestoreClocks()

この関数は、PSoC デバイスを CyPmSaveClocks() の呼び出し時に保存された設定に復帰させるために使用します。これは通常、低電力モード終了後に最初に呼び出す関数です。

C.3 CyPmAltAct()

この関数は、AltAct モードへの遷移を管理し、wakeupTime と wakeupSource という 2 つのパラメーターを持ちます。wakeupTime パラメーターは、タイマーの周波数を設定し、それをウェイクアップ ソースとしてマスク解除するために使用します。wakeupSource パラメーターは、非同期およびコンポーネント ベースのウェイクアップ ソースをマスク解除するために使用します。

注: 「割り込み」が AltAct ウェイクアップ ソースとして設定した場合、個々の割り込み成分をマスクすることはできません。そのうえ、割り込みコントローラーによってフィルタリングされる前に、パワーマネージャにはすべての生の割り込みが見えます。つまり、通常に割り込み信号に適用されるすべてのエッジ検出やイネーブル設定が無視されます。

C.4 CyPmSleep()

この関数は、スリープ モードへの遷移を管理し、wakeupTime と wakeupSource という 2 つのパラメーターを持ちます。wakeupTime パラメーターは、タイマーの周波数を設定し、それをウェイクアップ ソースとしてマスク解除するために使用します。wakeupSource パラメーターは、非同期およびコンポーネント ベースのウェイクアップ ソースをマスク解除するために使用します。

スリープ モードに入る前にクロックの状態が正しく保存されることを確実にするために、この関数の前に CyPmSaveClocks() を呼び出す必要があります。

C.5 CyPmHibernate() または CyPmHibernateEx()

CyPmHibernate() 関数はハイバネート モードへの遷移を管理します。パラメーターがなく、PICU と XRES をウェイクアップ ソースとして使用します。

CyPmHibernateEx() 関数は、ハイバネート モードからのウェイクアップ ソースを変更するためのパラメーターが 1 つあります。PICU 割り込み、コンパレータ 0、コンパレータ 1、コンパレータ 2 およびコンパレータ 3 出力に設定できます。これにより、ユーザーは正確なウェイクアップ ソースを指定し、異なる動作モードの必要に応じて容易に変更できます。基本的には、デザインは必要ないウェイクアップ ソースをマスクすることが可能です。

ハイバネート モードに入る前にクロックの状態が正しく保存されることを確実にするために、この関数の前に CyPmSaveClocks()を呼び出す必要があります。また、スリープ レギュレータを確実に安定させるために、ハイバネート モードからの復帰後少なくとも 20µs までは、PSoC デバイスを再び低電力モードに入れないようにしてください。

C.6 コンポーネントの低消費電力 API

ほとんどの PSoC Creator のコンポーネントは、コンポーネントを低消費電力状態にするための API 関数を持っています。Sleep 関数はコンポーネントの設定を保存して、Stop 関数を呼び出します。Stop と Sleep の間には、省電力の差はありません。

コンポーネントをウェイクアップ ソースとして使用しない限り、Sleep 関数は CyPmSleep() または CyPmHibernate() の呼び出しの前に呼び出す必要があります。Sleep 関数のフォーマットは次の通りです。

```
MyComponentName_Sleep();
```

Sleep 関数を持つすべてのコンポーネントは、コンポーネントを前の状態に復帰させる Wakeup 関数も持っています。Wakeup の関数のフォーマットは次の通りです。

```
MyComponentName_Wakeup();
```

Sleep と Stop 関数は、コンポーネントのアイドルまたはビジー ステータスをチェックしません。アクティブなコンポーネントがビジーではないことを確実にするため、コードにチェック処理を入れる必要があります。低消費電力動作の管理方法の詳細は、個々のコンポーネントのデータシートを参照してください。

C.7 直接レジスタ書き込み

電力モードの遷移を制御するためには、可能な限り API 関数を使用するべきですが、代わりにレジスタ書き込みを使用することもできます。ここで言及するレジスタは、電力モードの遷移に関連する最も一般的に使用されるものです。

C.7.1 PM_MODE_CSR

電力モード制御およびステータス レジスタは、PSoC 電力モードに関連するレジスタの中でおそらく最も重要なものです。27 ページの表 4 に示すように、このレジスタのビット[2:0]は PSoC デバイス全体のグローバル電力モードを制御します。これらのビットを読み出すと、PSoC デバイスが現在どのモードにあるかが分かります。

表 4. PM_MODE_CSR[2:0]の値

[2:0]の値	モード
000	アクティブ モード
001	AltAct モード
010	未対応
011	スリープ モード
100	ハイバネートモード
101	未対応
110	未対応
111	未対応

これらのビットをセットしても、それだけでは PSoC デバイス全体は自動的に低消費電力動作に設定されません。グローバル電力設定および PSoC 3 と PSoC 5LP サブシステムへの影響の詳細は、PSoC 3 と PSoC 5LP の TRM に記載されています。

注: 未対応の低消費電力モードやウェイクアップ ソースを使用すると、信頼できない動作につながる可能性があります。低消費電力モードに入るためには可能な限り標準的な API を使用してください。

C.7.2 PM_TW_CFGx

これらの 3 つのレジスタは、PSoC を低消費電力モードから復帰させるために使用するタイマーを有効にし、設定します。

C.7.3 PM_WAKEUP_CFGx

これらの 3 つのレジスタは、電力モードのウェイクアップ ソースをマスクします。個々のソースは、動的にマスクしたりマスク解除したりすることができ、異なる条件の下で異なるウェイクアップ ソースを可能にします。

C.7.4 PICUx_INTTYPEy

これらのレジスタは PICU を設定します。割り込みトリガー条件を設定するために、ピンごとに 1 つのレジスタがあります。

C.7.5 PICUx_INTSTAT

これらのレジスタは、ピン割り込みのステータスを保持します。ポートごとに 1 つのステータス レジスタがあり、ピンあたりに 1 ビットが対応します。

C.7.6 FASTCLK

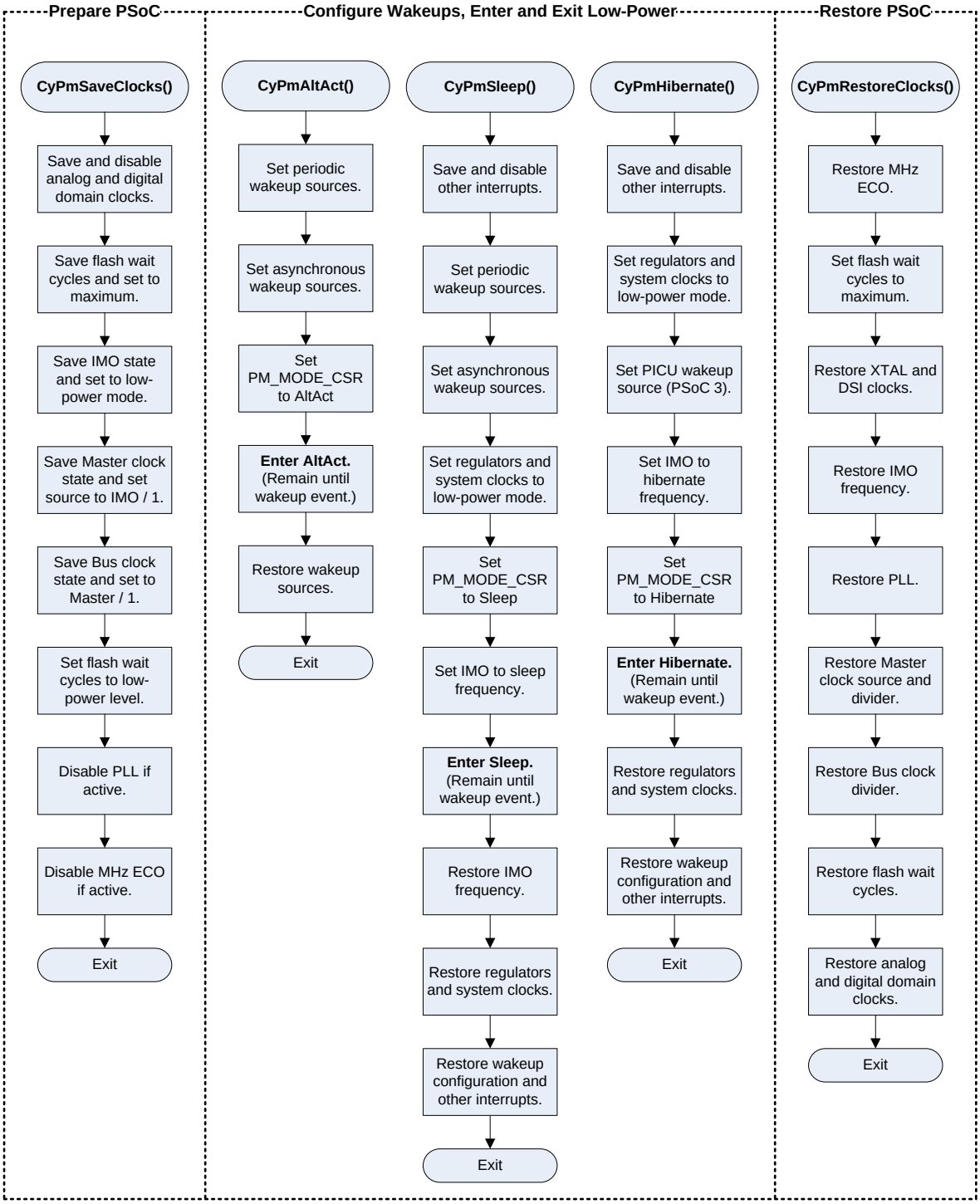
これらのレジスタは、IMO、マスター クロックおよび PLL を設定します。クロック分配は CLKDIST レジスタによって制御されます。

C.7.7 SLOWCLK

これらのレジスタは、内部低速発振器 (ILO) と 32kHz 水晶振動子を設定します。クロック分配は CLKDIST レジスタによって制御されます。

C.8 電力管理 API のフロー チャート

図 28. 電力管理機能のフロー チャート



C.9 電力管理 API レジスタ リファレンス

表 5 は、PSoC 3 と PSoC 5LP デバイスで用意されている非同期ウェイクアップ ソースの一覧です。

表 5. CyPm ウェイクアップ ソースのコンフィギュレーション

API	ウェイクアップ ソース	定義された wakeupSource マスク	影響を受けるレジスタ [ビット]
CyPmAltAct()	コンパレータ 0	PM_ALT_ACT_SRC_COMPARATOR0	PM_WAKEUP_CFG1 [0x01]
	コンパレータ 0	PM_ALT_ACT_SRC_COMPARATOR1	PM_WAKEUP_CFG1 [0x02]
	コンパレータ 0	PM_ALT_ACT_SRC_COMPARATOR2	PM_WAKEUP_CFG1 [0x04]
	コンパレータ 0	PM_ALT_ACT_SRC_COMPARATOR3	PM_WAKEUP_CFG1 [0x08]
	任意の割り込み	PM_ALT_ACT_SRC_INTERRUPT	PM_WAKEUP_CFG0 [0x01]
	PICU	PM_ALT_ACT_SRC_PICU	PM_WAKEUP_CFG0 [0x04]
	I^2C アドレス	PM_ALT_ACT_SRC_I2C	PM_WAKEUP_CFG0 [0x08]
	ブースト	PM_ALT_ACT_SRC_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	FTW	PM_ALT_ACT_SRC_FTW	PM_WAKEUP_CFG0 [0x40]
	CTW	PM_ALT_ACT_SRC_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_ALT_ACT_SRC_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD	PM_ALT_ACT_SRC_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmSleep()	コンパレータ 0	PM_SLEEP_SRC_COMPARATOR0	PM_WAKEUP_CFG1 [0x01]
	コンパレータ 1	PM_SLEEP_SRC_COMPARATOR1	PM_WAKEUP_CFG1 [0x02]
	コンパレータ 2	PM_SLEEP_SRC_COMPARATOR2	PM_WAKEUP_CFG1 [0x04]
	コンパレータ 3	PM_SLEEP_SRC_COMPARATOR3	PM_WAKEUP_CFG1 [0x08]
	PICU	PM_SLEEP_SRC_PICU	PM_WAKEUP_CFG0 [0x04]
	I^2C アドレス	PM_SLEEP_SRC_I2C	PM_WAKEUP_CFG0 [0x08]
	ブースト コンバーター	PM_SLEEP_SRC_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	CTW	PM_SLEEP_SRC_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_SLEEP_SRC_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD ガラス駆動	PM_SLEEP_SRC_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmHibernate()	PICU	該当なし	PM_WAKEUP_CFG0 [0x04]

表 6 は、PSoC 3 デバイスで用意されている周期的なウェイクアップ ソースの一覧です。

表 6. CyPm ウェイクアップ周期のコンフィギュレーション

API	ウェイクアップ ソース	定義された wakeupTime マスク	影響を受けるレジスタ [ビット]
CyPmAltAct() と CyPmSleep()1	OPPS	PM_ALT_ACT_TIME_ONE_PPS PM_SLEEP_TIME_ONE_PPS	PM_TW_CFG2 [0x20] – マスク解除 PM_TW_CFG2 [0x10] – イネーブル
	CTW 2ms	PM_ALT_ACT_TIME_CTW_2MS PM_SLEEP_TIME_CTW_2MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x01] – 2ms 周期
	CTW 4ms	PM_ALT_ACT_TIME_CTW_4MS PM_SLEEP_TIME_CTW_4MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x02] – 4ms 周期
	CTW 8ms	PM_ALT_ACT_TIME_CTW_8MS PM_SLEEP_TIME_CTW_8MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x03] – 8ms 周期
	CTW 16ms	PM_ALT_ACT_TIME_CTW_16MS PM_SLEEP_TIME_CTW_16MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x04] – 16ms 周期
	CTW 32ms	PM_ALT_ACT_TIME_CTW_32MS PM_SLEEP_TIME_CTW_32MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x05] – 32ms 周期
	CTW 64ms	PM_ALT_ACT_TIME_CTW_64MS PM_SLEEP_TIME_CTW_64MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x06] – 64ms 周期
	CTW 128ms	PM_ALT_ACT_TIME_CTW_128MS PM_SLEEP_TIME_CTW_128MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x07] – 128ms 周期
	CTW 256ms	PM_ALT_ACT_TIME_CTW_256MS PM_SLEEP_TIME_CTW_256MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x08] – 256ms 周期
	CTW 512ms	PM_ALT_ACT_TIME_CTW_512MS PM_SLEEP_TIME_CTW_512MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x09] – 512ms の周期
	CTW 1024ms	PM_ALT_ACT_TIME_CTW_1024MS PM_SLEEP_TIME_CTW_1024MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x0A] – 1024ms 周期
	CTW 2048ms	PM_ALT_ACT_TIME_CTW_2048MS PM_SLEEP_TIME_CTW_2048MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x0B] – 2048ms 周期
	CTW 4096ms	PM_ALT_ACT_TIME_CTW_4096MS PM_SLEEP_TIME_CTW_4096MS	PM_TW_CFG2 [0x08] – マスク解除 PM_TW_CFG2 [0x04] – イネーブル PM_TW_CFG1 [0x0C] – 4096ms 周期
	FTW 10µs～2.56ms (AltAct のみ)	PM_ALT_ACT_TIME_FTW(1-256)	PM_TW_CFG2 [0x02] – マスク解除 PM_TW_CFG2 [0x01] – イネーブル PM_TW_CFG0 [0x00～0xFF]周期
CyPmHibernate()	該当なし	該当なし	該当なし

[1] PSoC 5LP はスリープ タイマーまたは RTC コンポーネントの一部として CTW と OPPS のウェイクアップ ソースのみをサポートします。

改訂履歴

文書名: AN77900 - PSoC® 3 と PSoC 5LP の低電力モードおよび電力低減技術

文書番号: 001-97884

版	ECN	変更者	発行日	変更内容
**	4802471	HZEN	07/03/2015	これは英語版 001-77900 Rev. *C を翻訳した日本語版 001-97884 Rev. **です。
*A	6252503	SSAS	07/19/2018	これは英語版 001-77900 Rev. *G を翻訳した日本語版 001-97884 Rev. *A です。

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

製品

Arm® Cortex®マイクロコントローラー	cypress.com/arm
車載用	cypress.com/automotive
クロック & バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
モノのインターネット (IoT)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラー	cypress.com/mcu
PSoC	cypress.com/psoc
電源管理 IC	cypress.com/pmic
タッチ センシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス接続	cypress.com/wireless

PSoC®ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

サイプレス開発者コミュニティ

フォーラム | WICED IOT フォーラム | プロジェクト | ビデオ | ブログ | トレーニング | コンポーネント

テクニカル サポート

cypress.com/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2012-2018. 本書面は, Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社 (以下「Cypress」という。) に帰属する財産である。本書面 (本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア (以下「本ソフトウェア」という。) を含む) は, アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し,本段落で特に記載されているものを除き, その特許権,著作権,商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず, かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合, Cypress は, (1) 本ソフトウェアの著作権に基づき, (a) ソースコード形式で提供されている本ソフトウェアについて, Cypress ハードウェア製品と共に用いるためにのみ, かつ組織内部でのみ, 本ソフトウェアの修正及び複製を行うこと, 並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ, (直接又は再販売者及び販売代理店を介して間接のいずれかで) 本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること, 並びに (2) 本ソフトウェア (Cypress により提供され, 修正がなされていないもの) が抵触する Cypress の特許権のクレームに基づき, Cypress ハードウェア製品と共に用いるためにのみ, 本ソフトウェアの作成, 利用, 配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス (サブライセンスの権利を除く) を付与する。本ソフトウェアのその他の使用, 複製, 修正, 変換又はコンパイルを禁止する。

適用される法律により許される範囲内で, Cypress は, 本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても, 明示又は黙示をとわず, いかなる保証 (商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない) も行わない。 いかなるコンピューティングデバイスも絶対に安全ということはない。従って, Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず, Cypress は, Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて, 本書面に記載された製品には, エラッタと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり, 公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で, Cypress は, 別途通知することなく, 本書面を変更する権利を留保する。Cypress は, 本書面に記載のある, いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報 (あらゆるサンプルデザイン情報又はプログラムコードを含む) は, 参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計, プログラム, かつテストすることは, 本書面のユーザーの責任において行われるものとする。Cypress 製品は, 兵器, 兵器システム, 原子力施設, 生命維持装置若しくは生命維持システム, 蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム, 汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用, 又は装置若しくはシステムの不具合が人身傷害, 死亡若しくは物的損害を生じさせるようなその他の使用 (以下「本目的外使用」という。) のためには設計, 意図又は承認されていない。重要な構成部分とは, それの不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ, 若しくは本目的外使用に関連するいかなる請求, 損害又はその他の責任についても, Cypress はその全部又は一部をとわず一切の責任を負わず, かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求, 費用, 損害及びその他の責任 (人身傷害又は死亡に基づく請求を含む) から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, CapsSense, EZ-USB, F-RAM, 及び Traveo は, 米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは, cypress.com を参照すること。その他の名称及びブランドは, それぞれの権利者の財産として権利主張がなされている可能性がある。