16-Bit カウンタのデータシート Counter16 V 2.5

Copyright © 2002-2010 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ（バイト）		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	
CY8C29/27/24/22/21xxx、CY8C23x33、CYWUSB6953、CY7C64215、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01						
16-bit	2	0	0	93	0	1
CY8C26/25xxx						
16-bit	2	0	0	142	0	1

このユーザモジュールを使用したサンプルプロジェクトは、以下を参照してください。
www.cypress.com/psocexampleprojects。

特徴と概要

- 16-bit 汎用カウンタは、それぞれ 1 ， 2 ， 3 あるいは 4 つの PSoC ブロックを使用
- 48 MHz までのソース クロック レート
- カウント終了でカウント期間の自動リロード
- プログラマブルなパルス幅
- 連続したカウンタ動作を有効 / 無効にする入力
- カウンター致出力 *または* カウント終了での割り込みオプション

16-bit カウンタ ユーザ モジュールは、ダウンカウンターで構成され、その期間とパルス幅はプログラマブルです。クロックとイネーブル信号は、システムのタイムベースまたは外部ソースから選択できます。開始されると、カウンタは連続して動作し、カウント終了に到達した際に、ピリオドレジスタからその値をリロードします。各クロック周期で、カウンタは現在のカウントの値と比較レジスタに保存された値を比較します。各クロック周期で、カウンタは、「未満」または 「以下」のいずれかで、比較レジスタの値に対して、カウンタの値をテストします。コンパレータ出力は、ピンやその他のユーザ モジュールに配線できるロジック レベルを提供します。ほとんどの PSoC デバイス ファミリは、同じ方法でカウント終了出力を配線することも可能です。デバイスにこの機能がある場合は、デバイス エディタに表示されます。カウンタがカウント終了に到達した場合、またはコンパレータ (プライマリ) 出力がアサートされた場合に、割り込みをトリガするようプログラムできます。

Figure 1. カウンタブロック図 (ほとんどの PSoC デバイス)、データパス幅 n = 16 ビット

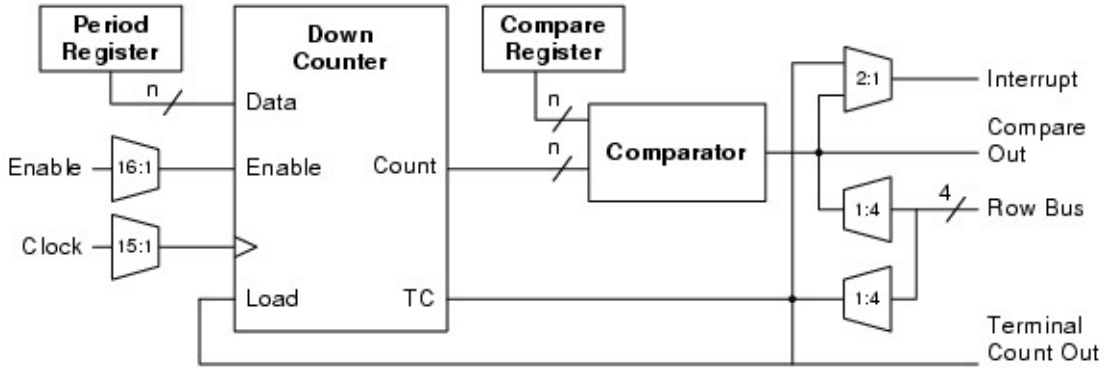
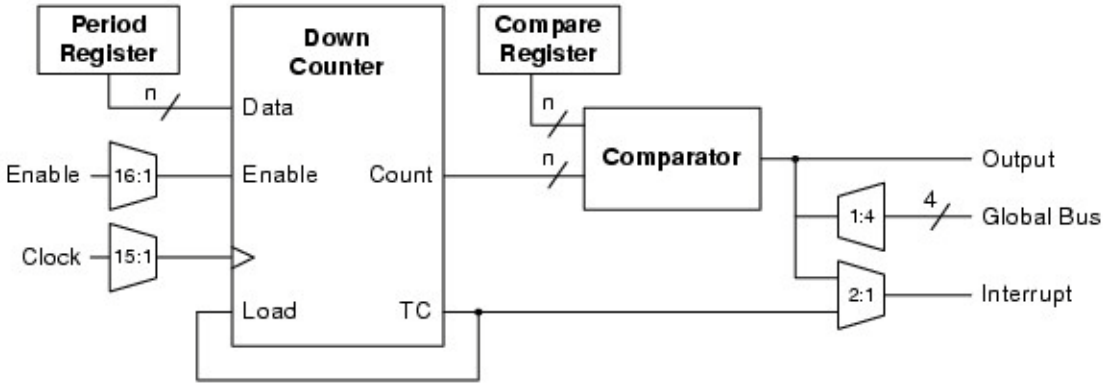


Figure 2. カウンタブロック図 (カウント終了出力のないデバイス)、データパス幅 n = 16 ビット



機能説明

カウンタ ユーザ モジュールは、 1 ~ 4 つのデジタル PSoC ブロックを採用し、それぞれのブロックは総分解能に対して 8 ビットずつ割り当てられます。8 ビット以上の買う運たーを作るには、連続した複数のブロックがリンクされ、その結果として内部キャリー、カウント終了あるいは比較信号が同期してつながれます。これは、ブロックからブロックへの 8-bit カウント、ピリオドおよび比較レジスタ (それぞれデータ レジスタ DR0、DR1、DR2) を連結し、必要な分解能を提供します。この方法で、8 ビットを超えるカウンタが、単一のモノリシック同期カウンタとして動作するのです。

カウンタ API は、C およびアセンブリから呼び出される関数を提供し、カウンタの操作の停止および開始ならびに、さまざまなデータ レジスタとの読み書きを実行します。データ レジスタ値は、デバイス エディタを使用することでも確定できます。開始されると、カウント レジスタは、アクティブな高イネーブル入力信号がアサートされる各クロック周期の立ち上がりエッジで減少されます。カウント レジスタがゼロに達すると、カウント終了に続く立ち上がりクロック エッジで、期間レジスタからカウントがリロードされます。

ピリオドレジスタは、いつでも新しい値で変更できます。カウンタが停止すると、ピリオドレジスタに値を書き込むとカウント レジスタの値も同時に変更されます。カウンタが実行されている間は、期間レジスタへの書き込みでは、カウント終了に続く次のリロードが発生するまで、新しい期間値で更新されません。カウント終了は、カウントがゼロになったときに到達するため、動作および出力信号の期間は、期間レジスタに保存されている値より 1 大きくなります。入力クロックの期間は、次の式で求められます。

Equation 1

$$OutputPeriod = (PeriodValue + 1)t_{CLK}$$

カウンタは、停止したときにその ロー出力をアサートします。実行中は、コンパレータが、出力信号のデューティサイクルを制御します。クロック周期ごとに、このコンパレータがカウント レジスタの値と比較レジスタの値をテストします。コンパレータは、デバイス エディタを使って選択されたオプションにしたがって、「未満」または 「以下」テストを実行します。カウンタは、比較が行われる期間に続く、クロックの立ち上がりエッジで、比較した値をアクティブハイの真偽値をアサートします。比較値と期間値の比が、出力波長のデューティサイクルを設定します。デューティサイクルの比は、次の式で計算できます。

Equation 2

$$DutyCycle = \begin{cases} \dfrac{CompareValue}{PeriodValue + 1}, \text{For } Less\ Than \text{ comparison} \\ \dfrac{CompareValue + 1}{PeriodValue + 1}, \text{For } Less\ Than\ Or\ Equal\ To \text{ comparison} \end{cases}$$

次の表は、期間レジスタ、比較レジスタ、比較操作の設定を基にした、特別の信号状態を要約しています。

Table 1. カウンタの特別出力信号状態

期間レジスタの値	比較タイプ	比較レジスタの値	パルス幅高タイマーと期間の比
0	考慮しない	> 0	1.0
0	≤	0	1.0
0	<	0	0.0
> 0	≤	0	1/(期間 + 1)
> 0	<	0	0.0
期間 = 比較	≤	期間 = 比較	1.0
期間 = 比較	<	期間 = 比較	期間 /(期間 + 1)
比較値 > 期間値	考慮しない	比較値 > 期間値	1.0

比較レジスタの値は、デバイス エディタを使って、あるいは API を使ってランタイム中に設定できます。比較レジスタのバッファリングは、期間レジスタがカウント終了前にカウント レジスタをバッファする方法の様には提供されません。このため、比較レジスタへの変更は、続くカウント終了ではなく、次のクロック周期における比較出力に影響します。これは、複数パルスを持つ期間を生成します。

CY8C29/27/24/22/21xxx デバイス ファミリでは、カウンタ ユーザ モジュールは、AUX 出力として、カウント終了信号を提供します。このアクティブハイの信号は、カウント終了に続いて期間レジスタからカウント レジスタがリロードされるクロック周期の立ち上がりエッジでアサートされます。

割り込みは、カウント終了、または比較が 「真」になる場合に発生するようにプログラムできます。コンパレータの出力は、出力信号の立ち上がりエッジで割り込みをトリガし、カウント終了は、出力信号の下降エッジの 1 . 5 クロック前に割り込みをトリガします。このオプションは、デバイス エディタを使って設定されます。割り込みは、カウンタ API を使って、ランタイム時に有効または無効にされます。グローバル割り込みは、カウンタの割り込みが実行される前に有効にする必要があります。

比較レジスタの値と現在のカウント値を組み合わせてカウンタの出力状態が決定されるため、比較レジスタの変更には注意が必要です。早すぎて ロー出力信号がアサートされたり、エラーが発生したりするのを防ぐため、比較レジスタは、カウント終了状態が割り込みを使って検出された後で変更してください。

[+] Feedback

デューティサイクルをより短い間隔で更新する必要がある用途では、カウンタの出力をその状態を CPU がポーリングするためにピンに迂回させることができます。ハイからローへの出力の移行が検出されたら、比較を更新できます。比較により、比較の 「真」状態が発生した場合は、出力が次のクロックでハイにアサートされます。

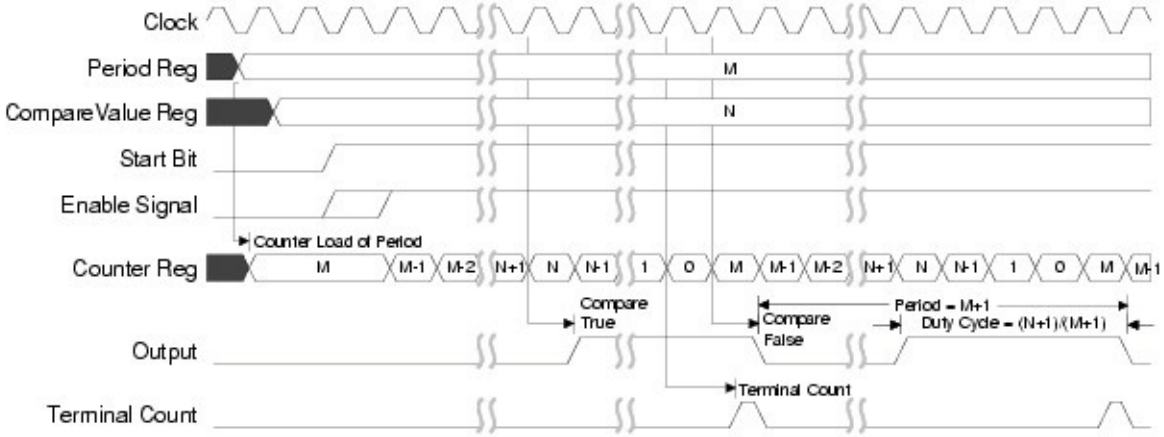
カウント レジスタの値は、注意して取得する必要があります。カウント レジスタを読み取ると、その内容が比較レジスタにラッチされます。これにより、出力デューティサイクルが変更されます。

カウント レジスタを随時読み取る必要がある場合には、ReadCounter() API 関数を呼び出すことができます。この関数は、一時的にクロックを無効にし、比較レジスタの内容を保存し、カウント レジスタと比較レジスタを読み取って、比較レジスタ、さらにクロックを復元します。副作用については、「アプリケーション プログラミング インターフェイス」に記載されている ReadCounter() 関数の説明を参照してください。

タイミング

カウンタ ユーザ モジュールの動作は、オンとオフを切り替えたり、PSoC デバイスのグローバル バス機能によりカウンタに迂回される外部ピンでクロッキングしたりできます。以下の図は、カウンタ ユーザ モジュールのタイミングを示しています。

Figure 3. カウンタのタイミング図



DC および AC の電気的特性

Table 2. CY8C26/25xxx デバイス ファミリにおける、カウンタの AC 電気的特性

パラメータ	標準値	制限	単位	条件および注意
最大出力周波数	--	48[1]	MHz	16-bit 幅、Vdd=5.0V[2]
最大出力周波数	--	24[1]	MHz	Vdd=5.0V および 48 MHz の入力クロック
	--	12[3]	MHz	Vdd=3.3V および 24 MHz の入力クロック

[+] Feedback

Table 3. CY8C29/27/24/22/21xxx デバイス ファミリにおける、カウンタの AC 電気的特性

パラメータ	標準値	制限	単位	条件および注意
最大出力周波数	--	48[1]	MHz	Vdd=5.0V[2]
最大出力周波数	--	24[1]	MHz	Vdd=5.0V および 48 MHz の入力クロック
	--	12[3]	MHz	Vdd=3.3V および 24 MHz の入力クロック

電気的特性に関する注意

1. 入力または出力が、グローバル バスを通して迂回される場合は、周波数は、最大 12 MHz に制限されます。
2. 提供されるイネーブル信号は常にハイになります。それ以外の場合は、24 MHz がリミットになります。
3. PSoC ブロックへの可能な最速のクロックは、3.3V の動作で 24 MHz です。

配置

カウンタは、8 ビットの分解能ごとに、1 つのデジタル PSoC ブロックを消費します。複数のブロックが割り当てられる場合は、最下位のバイト (LSB) から最上位のバイト (MSB) までブロック番号が増えるよう、デバイス エディタで順番に配置されます。各ブロックには、配置中または配置後、デバイス エディタで表示されるシンボリック名が与えられます。API は、ユーザが割り当てたインスタンス名とブロック名に対して、すべてのレジスタ名を確認し、ファイルを含め、API を通してカウンタ レジスタへの直接アクセスを提供します。さまざまなカウンタ ユーザ モジュールによって割り当てられたブロック名を、以下の表に示します。

Table 4. マップされる PSoC ブロックのシンボリック名

PSoC ブロック名	16-Bit カウンタ
1	CNTR16_LSB
2	CNTR16_MSB

パラメータおよびリソース

カウンタ ユーザ モジュールが、デバイス エディタを使って選択されて配置されると、以下のハラメータの値を選択したり、変更したりできます。

Clock (クロック)

Clock パラメータは、利用可能なソースのいずれかから選択されます。これらのソースには、48 MHz オシレータ (5.0V 動作のみ)、24 MHz システム クロックから分割されるより低い周波数 (24V1 および 24V2)、その他の PSoC ブロック、グローバル入出力を通して迂回される外部入力が含まれます。ブロックで外部デジタル クロックを使用している場合は、最高の精度およびスリープ動作を得るため、の入力同期をオフにします。

[+] Feedback

Enable (イネーブル)

Enable パラメータは、利用可能なソースのいずれかから選択されます。ハイ入力は、連続カウントを有効にし、ローのイネーブル信号はカウンタをリセットすることなくカウンタを向こうにします。

Output (出力)

Output パラメータは、無効にするか、4 つのグローバル出力信号のいずれかに迂回することができます。このパラメータは、PSoC デバイスの CY8C26/25xxx ファミリのみに適用されます。

CompareOut (比較出力)

CompareOut は、無効にするか (割り込み動作を妨害することなく)、未処理の出力バスに接続できます。このパラメータの設定にかかわらず、デジタル PSoC ブロックへの入力、およびアナログコラムのクロック選択マルチプレクサとして常に利用できます。PSoC デバイスの CY8C29/27/24/22/21xxx ファミリのみに適用されます。

TerminalCountOut (カウント終了出力)

TerminalCountOut は、AUX カウンタ出力です。このパラメータを使うと、無効にするか、未処理の出力バスに接続できます。PSoC デバイスの CY8C29/27/24/22/21xxx ファミリのみに適用されます。

Period (期間)

このパラメータは、カウンタの期間を設定します。許可される値は、0 ～ 2^n-1 の間です。ここで、n は、ビットのカウンタ幅です。期間は、期間レジスタにロードされます。カウンタの有効な出力波形期間は、期間カウント + 1 です。値は、API を使って変更できます。

CompareValue (比較値)

このパラメータは、比較レジスタで比較値を設定します。許可される値は、ゼロと期間値の間です。値は、API を使って変更できます。

CompareType (比較タイプ)

このパラメータは、上記の機能の説明に記載されているように、「未満」または 「以下」の比較関数タイプを設定します。

InterruptType (割り込みタイプ)

カウンタは、コンパレータの 「真」またはカウント終了で割り込みを生成しま s 。別のレジスタで、この割り込みを個別に有効にします。

ClockSync (クロック同期)

PSoC デバイスでは、デジタル ブロックは、システム クロックに加えて、クロック ソースを提供できます。デジタル クロック ソースは、リップル形式で連結することも可能です。これで、システム クロックに関するスキューが発生することになります。これらのスキューは、さまざまなデータパス最適化、特に、システム バスに適用される場合、CY8C29/27/24/22/21xxx PSoC デバイスでよりクリティカルになります。このパラメータは、制御クロック スキューに使用され、PSoC ブロック レジスタ値を読み書きする場合に、正しい動作を保証します。このパラメータの適切な値は、以下の表から決定してください。

ClockSync 値	使用
SysClk への同期	2 以上で除算される 24 MHz (SysClk) からの 派生クロック ソースでこの設定を使用します。例には、VC1、VC2、VC3 (VC3 が SysClk によって駆動される場合)、32KHz、SysClk ベースのソースを持つデジタル PSoC ブロックが含まれます。正しい同期が行われるよう、外部で生成されたクロック ソースもこの値を使用してください。
SysClk*2 への同期	結果の周波数が 48 MHz でない限り (言い換えると、すべての除算結果が 1 になる場合)、48 MHz (SysClk*2) ベースのクロックでは、この設定を使用します。
SysClk 指示の使用	24 MHz (SysClk/1) クロックが適切な場合に使用します。これは、同期を実際には実行しませんが、システム クロック自体への低スキュー アクセスを提供します。選択すると、このオプションは、上の Clock パラメータの設定を上書きします。組み合わせた全除算値の最終結果が 24 MHz 出力を生成する場合は、VC1、VC2、VC3、またはデジタル ブロックの代わりに常に使用してください。
Unsynchronized (非同期)	48 MHz (SysClk*2) 入力が選択される場合に使用します。 非同期入力が適切な場合に使用します。一般に、割り込み生成がカウンタの単独の用途である場合にのみ使用することを推奨します。この設定は、スリープ中でアクティブに維持されるブロックで必要です。

InvertEnable (反転有効)

このパラメータによって、イネーブル入力信号のセンスを決定します。「Normal (標準)」を選択すると、イネーブル入力がアクティブ高になります。「Invert (反転)」を選択すると、センスがアクティブ低として解釈されます。InvertEnable は、PSoC デバイスの CY8C29/27/24/22/21xxx ファミリにのみ適用されます。

割り込み生成の制御

PSoC Designer で、**[Enable interrupt generation control (割り込み生成の制御を有効にする)]** チェックボックスが選択されている場合は、さらに 2 つのパラメータが利用できるようになります。これは、**[Project(プロジェクト)] - > [Settings(設定)] - > [Chip Editor(チップエディタ)]** から利用できます。「割り込み生成の制御」は、オーバーレイ全体で複数のユーザ モジュールにより共有される割り込みとともに、複数のオーバーレイが使用される場合に重要です。

- 割り込み API
- IntDispatchMode

InterruptAPI

InterruptAPI パラメータを使うと、ユーザ モジュールの割り込みハンドラと割り込みベクトル テーブル エントリの状況に応じた生成が可能になります。「Enable (有効)」を選択すると、割り込みハンドラと割り込みベクトル テーブル エントリが生成されます。「Disable (無効)」を選択すると、割り込みハンドラと割り込みベクトル テーブル エントリがバイパスされます。1 つのブロック リソースが異なるオーバーレイで使用されるような、複数のオーバーレイを持つプロジェクトでは特に、割り込み API が生成されるかどうかを正しく選択してください。割り込み API の生成のみを選択すると、割り込みディスパッチ コードを生成する必要がなくなり、オーバーヘッドを軽減できます。

IntDispatchMode

同じブロック、異なるオーバーレイに存在する複数のユーザ モジュールによって共有される割り込みで、割り込みリクエストをどのように取り扱うかを指定します。ActiveStatus を選択すると、ファームウェアが、共有される割り込みリクエストに応答する前に、どちらのオーバーレイがアク

ティブであるかをテストします。このテストは、共有割り込みがリクエストされるたびに実行されます。これはレイテンシーを生むほか、共有割り込みリクエストに対応する非決定性のプロシージャを生成しますが、RAM は必要としません。OffsetPreCalc を選択すると、ファームウェアが、オーバーレイが最初にロードされるときだけ、共有割り込みリクエストのソースを計算するようになります。この計算は、割り込みレイテンシーを低減し、共有割り込みリクエストに応答する決定性のプロシージャを生成しますが、RAM のバイト消費が発生します。

アプリケーション・プログラミング インタフェース (API)

アプリケーション・プログラミング インタフェース (API) ルーチンは、設計者がモジュールをハイレベルで操作することを可能にします。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。

Note すべてのユーザーモジュール API の場合と同じように、API 関数を呼び出すことで A と X レジスタの値が変更されることがあります。API をコールした後でも A と X の全治を保持したいときは、API をコールするファンクションで AtoX の値を保持する必要があります。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、自動的にこの条件で処理されています。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も確認する必要があります。一部のユーザーモジュール 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。

以下に、Counter16: で提供されている変数を示します。

Counter16_PERIOD

説明：

デバイス エディタにおいて、Counter16 の Period フィールドで選択された値を示します。値は、0 ～ 65535 の範囲を持ちます。

Counter16_COMPARE_VALUE

説明：

デバイス エディタにおいて、Counter16 の PulseWidth フィールドで選択された値を示します。値は、0 ～ 65535 の範囲を持ちます。

以下に、Counter16: で提供されている API プログラミング ルーチンを示します。

Counter16_Start

説明：

Counter16 ユーザ モジュールを開始します。イネーブル入力が高の場合は、カウンタがカウントの減少を開始します。

C プロトタイプ：

```
void Counter16_Start(void);
```

アセンブリ：

```
lcall  Counter16_Start
```

パラメータ：

なし

[+] Feedback

戻り値：

なし

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_Stop

説明：

カウンタの動作を停止します。

C プロトタイプ：

```
void Counter16_Stop(void);
```

アセンブリ：

```
lcall  Counter16_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

出力がローにリセットされ、期間レジスタへの書き込みにより、カウンタ レジスタが新しい期間値で更新されます。この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_EnableInt

説明：

割り込みモード動作を有効にします。

C プロトタイプ：

```
void Counter16_EnableInt(void);
```

アセンブリ：

```
lcall  Counter16_EnableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_DisableInt

説明：

割り込みモード動作を無効にします。

C プロトタイプ：

```
void Counter16_DisableInt(void);
```

アセンブリ：

```
lcall  Counter16_DisableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。この関数、または今後この関数の実装によって、A および X レジスタか ゛変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_WritePeriod

説明：

期間値を期間レジスタに書き込みます。Counter16 が停止される、またはカウンタがゼロに達すると、期間値は、期間レジスタからカウンタ レジスタに直ちに転送されます。

C プロトタイプ：

```
void Counter16_WritePeriod(WORD wPeriod);
```

アセンブリ：

```
mov   X, [wPeriod]                  ;load the MSB, e.g. from RAM
mov   A, [wPeriod+1]                ;load the LSB
lcall  Counter16_WritePeriod
```

パラメータ：

wPeriod : wPeriod は、0 ー 65535 の値です。X レジスタでは MSB、A レジスタでは LSB が渡されます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_WriteCompareValue

説明：

CompareValue レジスタに比較値を書き込みます。

C プロトタイプ：

```
void Counter16_WriteCompareValue(WORD wCompareValue);
```

アセンブリ：

```
mov   X, [wCompareValue]
mov   A, [wCompareValue+1]
lcall  Counter16_WriteCompareValue
```

パラメータ：

wCompareValue : wCompareValue は、ゼロ一期間値の値です。X レジスタでは MSB、A レジスタでは LSB が渡されます。

戻り値：

なし

特殊作用：

カウンタがアクティブな間に CompareValue レジスタに書き込みが行われると、出力のデューティ サイクルが変更されます。これで、出力へのエラーが発生したり、気づかないうちに変更されたりします。この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

Counter16_wReadCompareValue

説明：

CompareValue レジスタを読み取ります。

C プロトタイプ：

```
WORD  Counter16_wReadCompareValue(void);
```

アセンブリ：

```
lcall  Counter16_wReadCompareValue
mov   [wCompareValue], X
mov   [wCompareValue+1], A
```

パラメータ：

wReadCompareValue : 比較レジスタから取得した 16-bit 値を返します。X レジスタでは MSB、A レジスタでは LSB が渡されます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数の責任です。

古いエイリアス：

wCounter16_ReadCompareValue - この名前は、PSoC Designer の今後のバージョンからは除外される場合があります。

Counter16_wReadCounter

説明：

比較レジスタを保持しながら、カウント レジスタ値 (ハードウェア DR0 レジスタ) を読み取ります。この関数は、カウントが実行中または停止されている間、カウント レジスタを読み取るために呼び出される場合があります。比較およびカウンタ レジスタが瞬時同じになっても、エラー割り込みが防止されます。ただし、重要な副作用があります (下記参照)。

C プロトタイプ：

```
WORD  Counter16_wReadCounter(void);
```

アセンブリ：

```
lcall  Counter16_wReadCounter
mov    [wCount], X
mov    [wCount+1], A
```

パラメータ：

なし

戻り値：

wReadCounter：カウント レジスタの wReadCounter 値。X レジスタでは MSB、A レジスタでは LSB が渡されます。

副作用：

カウンタ ユーザ モジュールが有効で、この関数が呼び出されたときにカウントを行っている場合は、ユーザ モジュールを短時間停止する必要があるため、一部のクロックが無視されることがあります (該当するカウント レジスタの減少が失われます)。この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

古いエイリアス：

wCounter16_ReadCounter - この名前は、PSoC Designer の今後のバージョンからは除外される場合があります。

サンプルファームウェア ソースコード

以下の例では、C とアセンブリ コード間の対応は比較的単純です。A レジスタでは LSB、X レジスタでは MBS を持つ簡単な 2 バイトの 「INT」パラメータの引渡しは、ユーザ モジュール API のアセンブラおよび C コンパイラによってパフォーマンスが最適化された形で使用されます。C コンパイラは、*Counter16.h* ファイルに #pragma fastcall 宣言を見つけると、スタックで引数をプッシュする代わりに、「INT」タイプでこのメカニズムを導入します。

以下のソースは、アセンブリ言語における API の使用を示しています。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;    This sample shows how to create a clock divider.  This specific
;    example divides the clock by 1000.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

include "m8c.inc"                     ; include the device interface
include "PSoCAPI.inc"                 ; include the API interface file

export _main

_main:

   mov   X, 03h                        ; set the period to 1000
   mov   A, E7h                        ;    i.e., 1000-1 = 999 = 0x03e7
   call  Counter16_WritePeriod

   mov   X, 01h                        ; Generate a 50% duty cycle
   mov   A, F3h                        ;    i.e., 500-1 = 499 = 0x01F3
   call  Counter16_WriteCompareValue

   call  Counter16_EnableInt           ; enable the Counter Interrupt
   M8C_EnableGInt                      ; enable global interrupts
   call  Counter16_Start               ; start to count when the enable
                                       ;    input is asserted
.terminate:
        jmp .terminate
```

C での同じコードは以下のようになります。

```
/*****************************************************************************
* Description:
*    This sample shows how to create a clock divider.  This specific
*    example divides the clock by 1000.
*****************************************************************************/

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

void main(void)
{
    Counter16_WritePeriod(999);            /* set the period to 1000   */
    Counter16_WriteCompareValue(499);      /* generate a 50 duty cycle */
    Counter16_EnableInt();                 /* disable the interrupt    */
    M8C_EnableGInt;                        /* enable global interrupts */
    Counter16_Start();                     /* start the counter        */
}
```

上の 2 つの場合、必要となる include ファイルは、必要な宣言を提供する *Counter16.h* のみです (プログラムおよびプロトタイプ)。アセンブリおよび C の include ファイルは、直接レジスタ アクセスがアプリケーションで必要な場合に、使用するシンボリック名を提供します。CY8C29/27/24/22/21xxx ユーザ モジュールでは、include ファイルは、より短い API 関数の一部のソースコードをインラインで拡張するために、マクロの定義も提供します。*Counter16.inc* および *Counter16.h* の代わりに、ファイル *PSoCAPI.inc* および *PSoCAPI.h* がそれぞれ使用される場合があります。これらのファイルには、PSoC Designer プロジェクトでユーザ モジュールを開始 (および配置) する include 文が含まれています。

設定レジスタ

16-bit カウンタは、2 つのデジタル PSoC ブロックを使用します。左から右への配置順では、CNTR16_LSB および CNTR16_MSB. という名前が付けられ、7 つのレジスタ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。以下の表では、定数としての 「パーソナリティ」値、あるいは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、およびその短い説明を示します。これらのレジスタのシンボリック名は、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。

Table 5. 関数レジスタ、バンク 1、CY8C26/25xxx

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	1	比較タイプ	割り込みタイプ	0	0	1
LSB	0	0	0	比較タイプ	0	0	0	1

Table 6. 関数レジスタ、バンク 1、CY8C29/27/24/22/21xxx

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	データ反転	0	1	比較タイプ	割り込みタイプ	0	0	1
LSB	0	BCEN	0	比較タイプ	0	0	0	1

BCEN は、比較出力を未処理のブロードキャスト バス ラインに出すかどうかゲートします。このビット フィールドは、ブロードキャスト ラインを直接設定して、デバイス エディタで設定されます。デバイス エディタで表示されるユーザ モジュール パラメータを通して設定されるデータの反転フラグは、イネーブル入力信号のセンスを制御します。CompareType フラグは、比較関数が 「以下」または 「未満」のどちらに設定されるかを示します。InterruptType フラグは、比較イベントまたはカウント終了のどちらで割り込みをトリガするかを決定します。CompareType と InterruptType は、このトピックで前述されているように、ユーザ モジュール パラメータを通して、デバイス エディタで直接設定されます。

Table 7. 入力レジスタ、バンク 1

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	0	0	1	1	Clock (クロック)			
LSB	Enable (イネーブル)				Clock (クロック)			

Enable は、16 ソースのいずれかからデータ入力を選択します。クロックは、15 ソースのいずれかから入力クロックを選択します。両ビット フィールドの値は、デバイス エディタで同じ名前を持つユーザ モジュール パラメータの設定によって決定されます。

Table 8. 出力レジスタ、バンク 1、CY8C29/27/24/22/21xxx

ブロック / ビット	7	6	5	4	3	2	1	0
MSB	AuxClk		AuxEnable	AuxSelect		OutEnable	OutputSelect	
LSB	AuxClk		0	0	0	0	0	0

デバイス エディタのユーザ モジュール 「ClockSync」パラメータは、AuxClk ビットの値を決定します。同じような名前が付いていますが、AuxEnable および AuxSelect は、OutEnable および OutSelect ビットフィールドにそれぞれ関係しています。AuxEnable および AuxSelect は、行出力バスのいずれかでカウント終了出力信号の駆動を許可し、デバイス エディタの配置ビューでグラフによる未処理のバス操作を使って制御されます。OutEnable は、比較出力が 2 つまたはグローバル出力バスのいずれかで駆動される場合に設定されます。OutputSelect は、比較出力からどちらのバスが駆動されるかを制御します。

Table 9. カウント レジスタ (DR0)、バンク 0

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

カウント レジスタは、イネーブル入力がアクティブであるクロック周期ごとに、1 減少する 16-bit ダウン カウント値です。その値は、カウント終了に続くクロック周期で、期間レジスタの内容からロードされます (ゼロ値)。Counter16 API を使って読み取ることができます。

Table 10. 期間レジスタ (DR1)、バンク 0

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

期間レジスタは、デバイス エディタまたは Counter16 API によって設定できる書き込み専用レジスタです。書き込まれると、ユーザ モジュールが API を通して無効になっている場合は、値がカウント レジスタに転送されます。その値は、カウント終了に続くクロック周期で、カウント レジスタに自動的にコピーされます。

Table 11. 比較レジスタ (DR2)、バンク 0

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

比較レジスタは、比較出力を生成するために、どのカウント レジスタがテストされるかに対する値を保持します。デバイス エディタおよび Counter16 API で設定できます。

Table 12. 制御レジスタ (CR0)、バンク 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	Start/Stop (開始 / 停止)

Start/Stop は、設定された場合に Counter24 を有効にし、クリアされた場合に無効にすることを示します。Counter16API を使って変更できます。

改訂履歴

バージョン	作成者	説明
2.5	TDU	クロックの説明に関する更新：ブロックで外部デジタル クロックを使用している場合は、最高の精度およびスリープ動作を得るため、未処理の入力同期がオフになります。

Note PSoC Designer 5.1 は、全ユーザ モジュール データシートのハージョンヒストリーを提供します。このセクションは、現在および以前のユーザ モジュール バージョン間の差をハイレベルに説明するものです。

Copyright © 2002-2010 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.