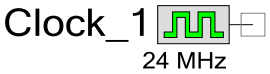
クロック
1.50

特長



- 新しいクロックをすばやく設定。
- システムまたはデザインワイドクロックを参照。
- クロック周波数の許容範囲を設定可能。

概要説明

クロックコンポーネントは 2 つの主要な機能を提供します。ローカルクロックを作成する手段と、デザインをシステムおよびデザインワイドクロックに接続する手段です。すべてのクロックはデザイン ワイド リソース (DWR) クロック エディタに表示されます。詳細については、PSoC Creator ヘルプ、クロック エディタ セクションを参照してください。

クロックは、さまざまな方法で定義することができます、例えば:

- 自動的に選択したクロックソースの周波数として
- ユーザが選択したクロックソースの周波数として
- ディバイダおよびユーザが選択したクロックソースとして

希望する周波数が指定されている場合は、PSoC Creator が、最も正確な周波数を生成するように、ディバイダを自動的に選択します。このとき、PSoC Creator は全てのシステムとデザインワイドクロックの中から、最も正確な周波数を生成するソースとディバイダの組み合わせを選び出します。

外観

クロックコンポーネント波形記号の色は、クロックのドメイン(DWR Clock Editor に記載の通り)に基づいて以下のように変化します。

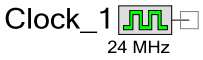
- デジタル-波形の色はデジタルワイヤと同じ色で、黒で縁取りしています
- アナログ-波形の色はアナログワイヤと同じ色で、黒で縁取りしています
- 不定-波形の色は白で、縁取りはありません

入出力接続

ここでは、クロックのさまざまな入出力接続について説明します。I/O リストのアスタリスク (*) は、I/O が、その I/O の説明でリストされている条件において、シンボルに隠れている可能性があることを示します。

クロック - 出力

クロックはクロック信号にアクセスできる標準出力ターミナルがあります。



デジタル ドメイン - 出力 *

イネーブルの場合は、このオプショナルな出力はアナログクロックからデジタルドメイン出力へのアクセスを提供します。Configure ダイアログの **Advanced** タブにあるオプションからこの出力をイネーブルにします。



コンポーネント パラメータ

クロックをデザイン上にドラッグし、ダブルクリックして Configure ダイアログを開きます。

注意 デザイン上に追加する任意のローカルクロックにおいて、DWR クロックエディタは、デフォルトでイネーブルになっている「リセット時にスタート」オプションがあります。消費電力を削減するなど、場合によっては、クロックをプログラムによりコントロールしたい場合があります。このような場合、「リセット時にスタート」オプションを選択解除し、Clock_Start() 関数をコードに挿入してください。詳細については、PSoC Creator ヘルプ、クロック エディタ セクションと API セクションを参照してください。

クロック タブの構成

「クロックの構成」タブは「クロックタイプ」と「ソース」パラメータがあります。選択に基づいて、このタブには次の図のように、さまざまな他のパラメータが含まれます。

図 1 Clock Type (クロックのタイプ): New / Source: <Auto>

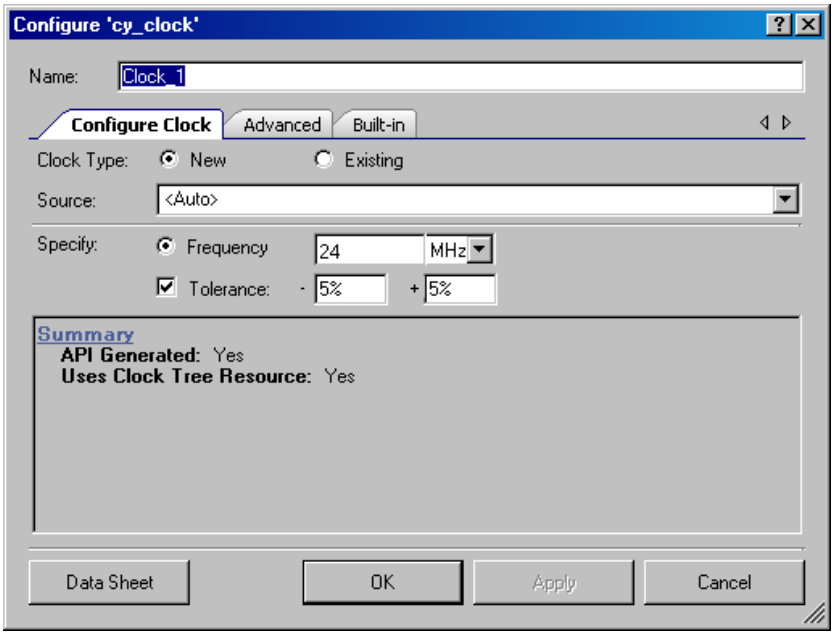


図 2 Clock Type (クロックのタイプ): New / Source: 特定のクロック

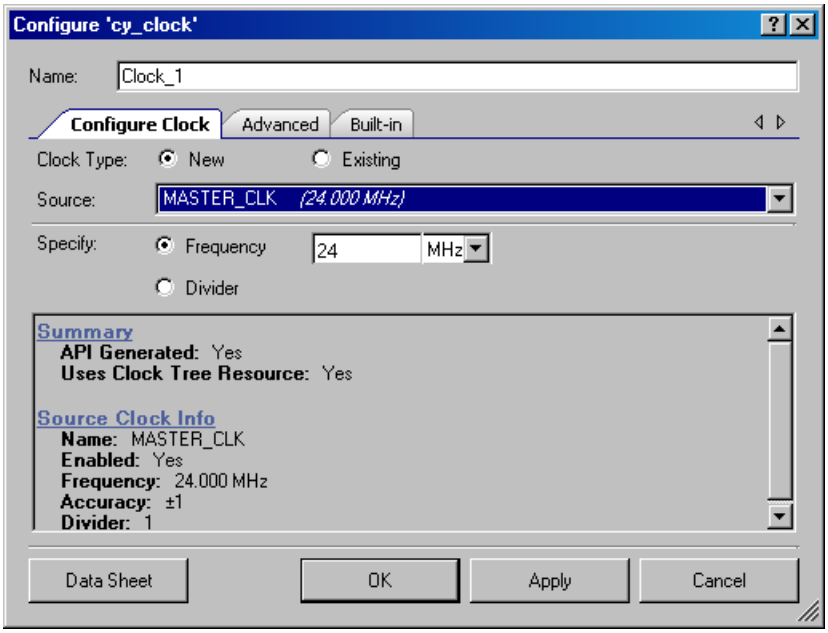
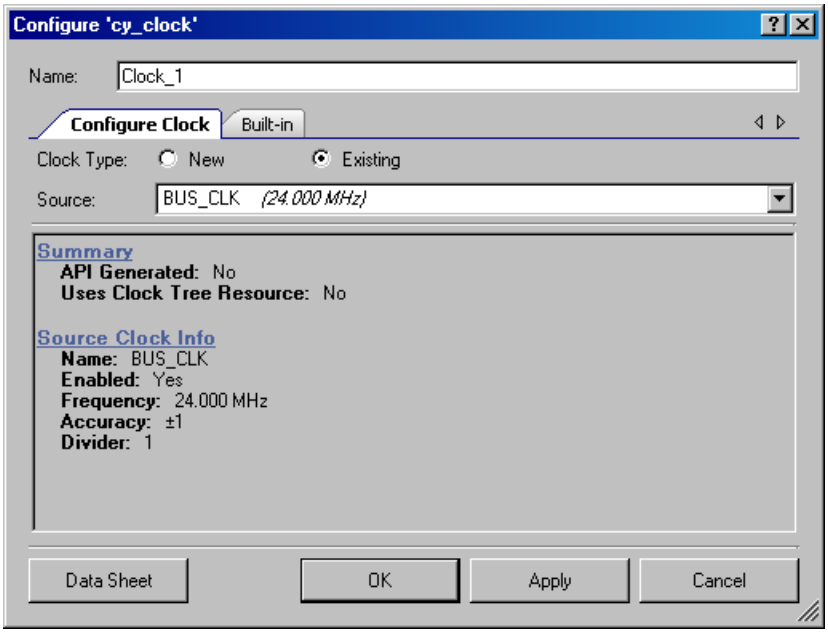


図 3 Clock Type (クロックのタイプ): 既存

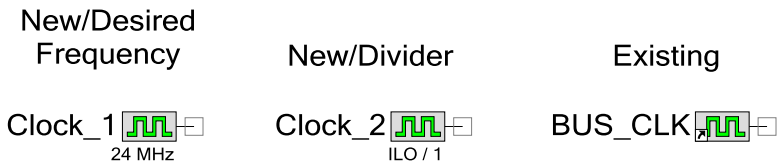


次のセクションでは、クロック コンポーネントのパラメータを説明します。

Clock Type (クロックのタイプ)

クロックタイプは 2 つあります: 新規および既存。新規クロックの場合、PSoC Creator を使うか、または<Auto>を選択して PSoC Creator に選ばせて、クロック**ソース**を指定できます。<Auto>を選択すると、特定の**周波数**およびオプションで**許容範囲**も入力できます。ソースを指定する場合は、**周波数**を指定するか、または**ディバイダ**を選択します。既存のクロックの場合、クロック**「ソース」**のみを選択できます。

異なる構成の場合、クロック シンボルは図面で異なる様式で表示されます。下記の例を参照してください。



"新規"として構成したクロックコンポーネントは、デバイス内のクロックリソースを消費し、そこから API を生成します。システムまたはデザインワイドクロックに"既存"として構成したクロックコンポーネントは、デバイス上の物理的なリソースを消費せず、そこから API は生成しません。その代わり、選択したシステムまたはデザイン全体のクロックが使用されます。

Source (ソース)

PSoC Creator は、<Auto> (デフォルト)を選択すると、分周時に最も正確な周波数を生成するように、利用可能なソースクロックを自動的に配置します。ソースが <Auto> のクロックは、希望の周波数のみを入力する場合があります。オプションとして、許容範囲を提供することもできます。

システムまたは、デザイン ワイド クロックを与えられたリストから選択してください。これは、PSoC Creator にそのクロックのソースとしての使用を強制するためです。

周波数

希望する周波数と単位を入力します(デフォルト = 24 MHz). PSoC Creator は、希望する周波数にできるだけ近い周波数のクロック信号を生成するようにディバイダを計算します。

許容範囲

クロックソースとして<Auto> を選択している場合は、クロックの希望する許容範囲を入力することが可能です(デフォルトは +/- 5%)。PSoC Creator は結果として生じたクロックの精度が、指定した許容範囲内であることを確認し、そうでない場合は警告を発します。クロックの許容範囲はパーセントとして指定します (**注** ppm を入力すると、入力した値が対応するパーセント値に変換されます。) 目的の許容範囲幅がない場合は、許容範囲の隣にあるチェックボックスにチェックしないでおくとこのクロックに対する警告は生成されません。

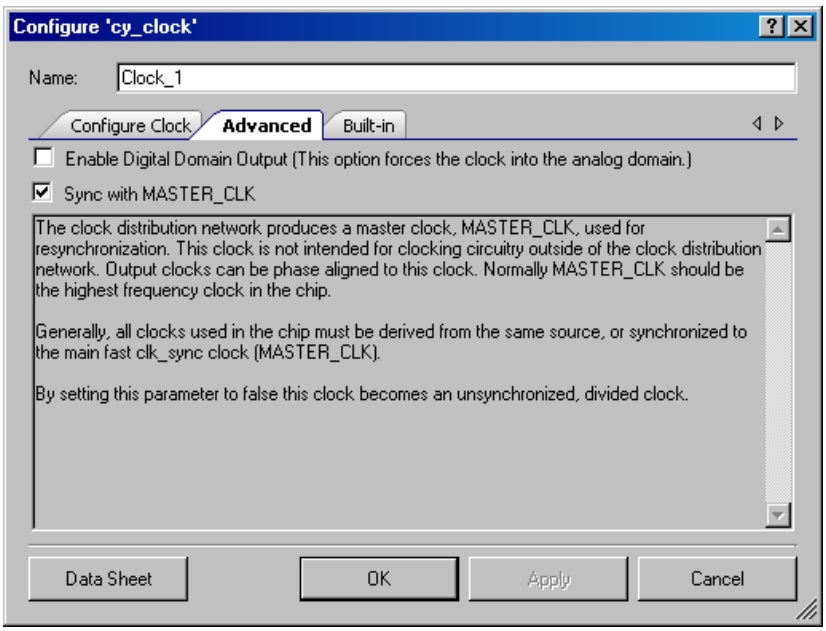
Divider (ディバイダ-分周器)

特定の**ソースクロック**を選択する場合は、**ディバイダ**の明確な値を入力することができます。さもないと、**ソースクロック**を <Auto>に設定したままだと、**ディバイダ** オプションは利用できません (デフォルト)。

ディバイダ オプションを選択すると、**許容範囲**オプションは利用できません。

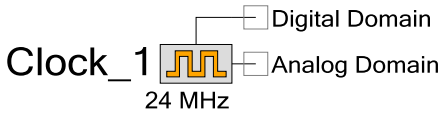
Advanced Tab(詳細設定タブ)

Advanced タブにはパラメータが 2 つあります。



デジタルドメイン出力のイネーブル

このオプションをチェックすると (デフォルトはチェックされていない) 、再同期クロックとしてメインデジタル同期クロックを使用するようにアナログクロックのバージョンにターミナルを追加します。使用した場合、クロックはアナログ ドメイン用に強制されますが、新しく追加されたターミナルはデジタル ドメイン内にあります。



MASTER_CLK での同期化

チェックすると(デフォルト=チェック済)クロックが Master クロックに同期します。さもないと、クロックは同期しません。

配置とリソース

リソースの使用は構成と接続形態によって変わります。

- 「既存」として構成されたクロックコンポーネントはチップ内のリソースを消費しません。
- 「新規」として構成されたクロックコンポーネントは一つのシングル クロックリソースを消費します。PSoC Creator は自動的にクロックがデジタルまたはアナログペリフェラルに接続するか、そして必要に応じてデジタル クロックまたはアナログ クロックを消費するかを見つけます。

	デジタル ブロック					API メモリ（バイト）		
アナログ ブロック	データパス	マクロセル	ステータスレジスタ	コントロールレジスタ	Counter7	フラッシュ	RAM	ピン (外部入出力ごと)
該当なし	該当なし	該当なし	該当なし	該当なし	該当なし	698	0	該当なし

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 インターフェース (API) ルーチンにより、ソフトウェアを使用してコンポーネントを設定できます。次の表は、各関数へのインターフェースとその説明を示しています。その次のセクションでは、各関数について詳しく説明します。

デフォルトでは、PSoC Creator はインスタンス名「Clock_1」をデザイン上のコンポーネントの最初のインスタンスに割り当てます。コンポーネントのインスタンス名は、識別子の文法ルールに従って固有の名前に変更できます。インスタンス名は、すべてのグローバル関数名、変数名、定数名のプリフィックスになります。読みやすいように、下表では「Clock」というインスタンス名を使用しています。

注意 構成ダイアログで**クロックタイプ**を「既存」にコンフィギュレーションされるローカルクロックは、API を生成しません。

機能	説明
Clock_Start	クロックをイネーブルにします。
Clock_Stop	クロックをディスエーブルにします。
Clock_StopBlock	クロックをディスエーブルにし、クロックがディスエーブルになるまで待ちます。
Clock_StandbyPower	スタンバイ (代替アクティブ) 操作モードの電源を選択します。
Clock_SetDivider	クロックのディバイダを設定し、クロック ディバイダを即座にリスタートします。
Clock_SetDividerRegister	クロックのディバイダを設定し、オプションとして、クロック ディバイダを即座にリスタートします。
Clock_SetDividerValue	クロックのディバイダを設定し、クロック ディバイダを即座にリスタートします。
Clock_GetDividerRegister	クロック ディバイダ レジスタ値を取得します。
Clock_SetMode	クロックの操作モードをコントロールするフラグを設定します。
Clock_SetModeRegister	クロックの操作モードをコントロールするフラグを設定します。
Clock_GetModeRegister	クロックのモードレジスタ値を取得します。
Clock_ClearModeRegister	クロックの操作モードをコントロールするフラグをクリアします。
Clock_SetSource	クロックのソースを設定します。
Clock_SetSourceRegister	クロックのソースを設定します。
Clock_GetSourceRegister	クロックのソースを取得します。
Clock_SetPhase	アナログ クロックの位相遅延を設定します (アナログ クロックにのみ生成) 。
Clock_SetPhaseRegister	アナログ クロックの位相遅延を設定します (アナログ クロックにのみ生成) 。
Clock_SetPhaseValue	アナログ クロックの位相遅延を設定します (アナログ クロックにのみ生成) 。
Clock_GetPhaseRegister	アナログ クロックの位相遅延値を取得します (アナログ クロックにのみ生成) 。

void Clock_Start(void)

説明: クロックをスタートします。

注意起動時、DWR Clock Editor において「リセット時にスタート」オプションがイネーブルの場合、クロックは既に稼働していることがあります。

パラメータ: 無効。

戻り値: 無効。

副作用: クロックはイネーブルです。

void Clock_Stop(void)

説明: クロックを停止し、ただちに戻ります。この API はソースクロックが稼働していることを要求しませんが、ハードウェアが実際にディスエーブルになる前に戻ることがあります。この関数を呼び出してからクロックの設定を変更すると、起動したときにクロックに問題が生じることがあります。クロックに問題が生じないようにするには、Clock_StopBlock() 関数を使用します。

パラメータ: 無効。

戻り値: 無効。

副作用: クロックはディスエーブルです。出力はロジック 0 です。

void Clock_StopBlock(void)

説明: クロックを停止し、ハードウェアが実際にディスエーブルになるまで待ってから戻ります。これにより、クロックが決して切り捨てられません (サイクルのHIGHの部分はクロックがディスエーブルになり、API が戻る前に中断します)。停止したクロックはディスエーブルにすることができないので、ソースクロックが稼働していないと、この API は絶対に戻りません。

パラメータ: 無効。

戻り値: 無効。

副作用: クロックはディスエーブルです。出力はロジック 0 です。

注 Clock_StopBlock() API は PSoC3 ES2 および PSoC5 ES1 ではサポートされていないので生成されません。

void Clock_StandbyPower(uint8 state)

説明: スタンバイ (代替アクティブ) 操作モードの電源を選択します。

パラメータ: uint8 state: 代替アクティブモード中はクロックをディスエーブルにするために0にし、イネーブルにするには0以外にします。

戻り値: 無効。

副作用: なし

void Clock_SetDivider(uint16 clkDivider)

説明: ディバイダを修正し、そのため周波数も修正されます。クロックディバイダのレジスタがゼロに設定された場合、あるいはゼロから変更された場合、モードビットを変更するために、クロックは一時的にディスエーブルになります。Clock_SetDivider() を呼び出したときにクロックをイネーブルにすると、ソースクロックは稼働している必要があります。現在のクロック サイクルは切り捨てられ、新しい分周値が直ちに有効になります。

パラメータ: uint16 clkDivider: ディバイダ レジスタ値 (0-65、535)。この値はディバイダ値では「ありません」。クロックハードウェアはclkDivider に1を加えたもので分周します。たとえば、2分周するためには、このパラメータを1に設定します。

戻り値: 無効。

副作用: なし

void Clock_SetDividerRegister(uint16 clkDivider, uint8 reset)

説明: ディバイダを修正し、そのため周波数も修正されます。クロックディバイダのレジスタがゼロに設定された場合、あるいはゼロから変更された場合、モードビットを変更するために、クロックは一時的にディスエーブルになります。Clock_SetDivider() を呼び出したときにクロックをイネーブルにするときは、ソース クロックは稼働している必要があります。

パラメータ: uint16 clkDivider: ディバイダ レジスタ値 (0-65、535)。この値はディバイダ値では「ありません」。クロック ハードウェアは clkDivider に1を加えたもので分周します。たとえば、2分周するためには、このパラメータを1に設定します。

uint8 reset: ゼロ以外の場合、クロック ディバイダがリスタートし、現在のクロックサイクルは切り捨てられ、新しい分周値が直ちにイネーブルになります。ゼロの場合、新しい分周値が現在のクロックサイクルの終わりにイネーブルになります。

戻り値: 無効。

副作用: なし

void Clock_SetDividerValue(uint16 clkDivider)

説明: ディバイダを修正し、そのため周波数も修正されます。クロックディバイダのレジスタがゼロに設定された場合、あるいはゼロから変更された場合、モードビットを変更するために、クロックは一時的にディスエーブルになります。Clock_SetDivider() を呼び出したときにクロックをイネーブルにするときは、ソースクロックは稼働している必要があります。現在のクロックサイクルは切り捨てられ、新しい分周値が直ちにイネーブルになります。

パラメータ: uint16 clkDivider: 分割値 (1-65535) またはゼロ。clkDivider がゼロの場合、クロックは 65,536 に分周されます。
これと Clock_SetDivider() の違いは、+1 要素を考慮する必要がないことです。

戻り値: 無効。

副作用: なし

uint16 Clock_GetDividerRegister(void)

説明: クロック ディバイダ レジスタ値を取得します。

パラメータ: 無効。

戻り値: クロックを割る数から1を引いた値。例えば、クロックを2で割るように設定している場合、返される値は1になります。

副作用: なし

void Clock_SetMode(uint8 clkMode)

説明: クロックの操作モードをコントロールするフラグを設定します。この関数はフラグを 0 から 1 に変更するだけです。すでに 1 であるフラグは変更されません。フラグをクリアするにはClock_ClearModeRegister() 関数を使用します。モードを変更する前にクロックをディスエーブルにする必要があります。

パラメータ: uint8 clkMode: 設定するビットを含むビットマスク。PSoC 3 およびPSoC 5では、clkMode は以下のオプショナルビットまたはそれらの集合のセットです。

- CYCLK_EARLY: 初期位相モードをイネーブルにします。ディバイダのカウンタが分周値の半分に達すると、出力クロックの立ち上がりエッジが生じます。
- CYCLK_DUTY: 50%デューティ比出力をイネーブルにします。イネーブルにした場合、出力クロックは周期の半分の間アサートされます。ディスエーブルの場合、出力クロックはソースクロックの1周期の間アサートされます。
- CYCLK_SYNC: 出力のマスタークロックへの同期をイネーブルにします。すべての同期クロックに対してイネーブルにする必要があります。

このクロックのモードの設定については、テクニカル リファレンス マニュアルを参照してください。特にCLKDIST.DCFG.CFG2 レジスタを参照してください。

戻り値: 無効。

副作用: なし

void Clock_SetModeRegister(uint8 clkMode)

説明: Clock_SetMode() と同じです。クロックの操作モードをコントロールするフラグを設定します。この関数はフラグを 0 から 1 に変更するだけです。すでに 1 であるフラグは変更されません。フラグをクリアするには Clock_ClearModeRegister() 関数を使用します。モードを変更する前にクロックをディスエーブルにする必要があります。

パラメータ: uint8 clkMode: 設定するビットを含むビット マスク。次のオプションビットである Ored がまとまったセットになります。

- CYCLK_EARLY: 初期位相モードをイネーブルにします。ディバイダのカウンタが分周値の半分に達すると、出力クロックの立ち上がりエッジが生じます。
- CYCLK_DUTY: 50% デューティ比出力をイネーブルにします。イネーブルにした場合、出力クロックは周期の半分の間アサートされます。ディスエーブルの場合、出力クロックはソースクロックの1周期の間アサートされます。
- CYCLK_SYNC: 出力のマスタークロックへの同期をイネーブルにします。すべての同期クロックに対してイネーブルにする必要があります。

このクロックのモードの設定については、テクニカル リファレンス マニュアルを参照してください。特に CLKDIST.DCFG.CFG2 レジスタを参照してください。

戻り値: 無効。

副作用: なし

uint8 Clock_GetModeRegister(void)

説明: クロックのモードレジスタ値を取得します。

パラメータ: void

戻り値: イネーブルになったモードビットを表すビットマスク。モードビットに関する詳細については、Clock_SetModeRegister() および Clock_ClearMode() Register をご覧ください。

副作用: なし

void Clock_ClearModeRegister(uint8 clkMode)

説明: クロックの操作モードをコントロールするフラグをクリアします。この関数はフラグを 1 から 0 に変更するだけです。すでに 0 であるフラグは変更されません。フラグをクリアするには Clock_ClearModeRegister() 関数を使用します。モードを変更する前にクロックをディスエーブルにする必要があります。

パラメータ: uint8 clkMode: クリアするビットを含むビットマスク。次のオプションビットである Ored がまとまったセットになります。

- CYCLK_EARLY: 初期位相モードをイネーブルにします。ディバイダのカウンタが分周値の半分に達すると、出力クロックの立ち上がりエッジが生じます。
- CYCLK_DUTY: 50%デューティ比出力をイネーブルにします。イネーブルにした場合、出力クロックは周期の半分の間アサートされます。ディスエーブルの場合、出力クロックはソースクロックの1周期の間アサートされます。
- CYCLK_SYNC: 出力のマスタークロックへの同期をイネーブルにします。すべての同期クロックに対してイネーブルにする必要があります。

このクロックのモードの設定については、テクニカルリファレンスマニュアルを参照してください。特に CLKDIST.DCFG.CFG2 レジスタを参照してください。

戻り値: 無効。

副作用: なし

void Clock_SetSource(uint8 clkSource)

説明: クロックの入力ソースを設定します。ソースを変更する前にクロックをディスエーブルにする必要があります。新旧のクロック ソースは稼働している必要があります。

パラメータ: uint8 clkSource: 次の入力ソースの 1 つである必要があります。

- CYCLK_SRC_SEL_SYNC_DIG: 位相遅延したマスタークロック。
- CYCLK_SRC_SEL_IMO: 内部メイン発振器。
- CYCLK_SRC_SEL_XTALM: 外部 4～33 MHz 水晶発振器。
- CYCLK_SRC_SEL_ILO: 内部低速発振器。
- CYCLK_SRC_SEL_PLL: 位相ロックループ出力。
- CYCLK_SRC_SEL_XTALK: 外部 32.768 kHz 水晶発振器。
- CYCLK_SRC_SEL_DSI_G: DSI グローバル入力信号。
- CYCLK_SRC_SEL_DSI_D: DSI デジタル入力信号。
- CYCLK_SRC_SEL_DSI_A: DSI アナログ入力信号。

クロック ソースの詳細については、テクニカル リファレンス マニュアルを参照してください。

戻り値: 無効。

副作用: なし

void Clock_SetSourceRegister(uint8 clkSource)

説明: Clock_SetSource() と同じクロックの入力ソースを設定します。ソースを変更する前にクロックをディスエーブルにする必要があります。新旧のクロック ソースは稼働している必要があります。

パラメータ: uint8 clkSource: 次の入力ソースの 1 つである必要があります。

- CYCLK_SRC_SEL_SYNC_DIG: 位相遅延したマスタークロック。
- CYCLK_SRC_SEL_IMO: 内部メイン発振器。
- CYCLK_SRC_SEL_XTALM: 外部 4～33 MHz 水晶発振器。
- CYCLK_SRC_SEL_ILO: 内部低速発振器。
- CYCLK_SRC_SEL_PLL: 位相ロックループ出力。
- CYCLK_SRC_SEL_XTALK: 外部 32.768 kHz 水晶発振器。
- CYCLK_SRC_SEL_DSI_G: DSI グローバル入力信号。
- CYCLK_SRC_SEL_DSI_D/CYCLK_SRC_SEL_DSI_A: DSI入力信号。

クロック ソースの詳細については、テクニカル リファレンス マニュアルを参照してください。

戻り値: 無効。

副作用: なし

uint8 Clock_GetSource(void)

説明: クロックの入力ソースを取得します。

パラメータ: 無効。

戻り値: クロックの入力ソースです。詳細は Clock_SetSourceRegister() を参照してください。

副作用: なし

void Clock_SetPhase(uint8 clkPhase)

説明: アナログクロックの位相遅延を設定する。この関数はアナログクロックでのみ利用できます。問題を避けるために、位相遅延を変更する前に、クロックをディスエーブルにする必要があります。

パラメータ: uint8 clkPhase: 1.0-ns単位でクロックの位相遅延を設定します。clkPhase は、インクルーシブに 1～11 でなければなりません。0 を含むその他の値は、クロックをディスエーブルにします。

clkPhase 値	PSoC 3 ES2 以前	PSoC 3 ES3 およびそれ以降、PSoC 5
0	クロック ディスエーブル	クロック ディスエーブル
1	2.5 ns	0.0 ns
2	3.5 ns	1.0 ns
3	4.5 ns	2.0 ns
4	5.5 ns	3.0 ns
5	6.5 ns	4.0 ns
6	7.5 ns	5.0 ns
7	8.5 ns	6.0 ns
8	9.5 ns	7.0 ns
9	10.5 ns	8.0 ns
10	11.5 ns	9.0 ns
11	12.5 ns	10.0 ns
12-15	クロック ディスエーブル	クロック ディスエーブル

戻り値: 無効。

副作用: なし

void Clock_SetPhaseRegister(uint8 clkPhase)

説明: Clock_SetPhase() と同じです。アナログクロックの位相遅延を設定する。この関数はアナログクロックでのみ利用できます。問題を避けるために、位相遅延を変更する前に、クロックをディスエーブルにする必要があります。

パラメータ: uint8 clkPhase: 1.0-ns単位でクロックの位相遅延を設定します。clkPhase は、インクルーシブに1～11 でなければなりません。0 を含むその他の値は、クロックをディスエーブルにします。

clkPhase 値	PSoC 3 ES2 以前	PSoC 3 ES3 およびそれ以降、PSoC 5
0	クロック ディスエーブル	クロック ディスエーブル
1	2.5 ns	0.0 ns
2	3.5 ns	1.0 ns
3	4.5 ns	2.0 ns
4	5.5 ns	3.0 ns
5	6.5 ns	4.0 ns
6	7.5 ns	5.0 ns
7	8.5 ns	6.0 ns
8	9.5 ns	7.0 ns
9	10.5 ns	8.0 ns
10	11.5 ns	9.0 ns
11	12.5 ns	10.0 ns
12-15	クロック ディスエーブル	クロック ディスエーブル

戻り値: 無効。

副作用: なし

void Clock_SetPhaseValue(uint8 clkPhase)

説明: アナログクロックの位相遅延を設定する。この関数はアナログクロックでのみ利用できます。問題を避けるために、位相遅延を変更する前に、クロックをディスエーブルにする必要があります。Clock_SetPhase() と同じですが、Clock_SetPhaseValue() は値に 1 を加算し、次に Clock_SetPhaseRegister() を呼び出します。

パラメータ: uint8 clkPhase: 1.0-ns 単位でクロックの位相遅延を設定します。clkPhase は、インクルーシブに 0～10でなければなりません。その他の値は、クロックをディスエーブルにします。

clkPhase 値	PSoC 3 ES2 以前	PSoC 3 ES3 およびそれ以降、PSoC 5
0	2.5 ns	0.0 ns
1	3.5 ns	1.0 ns
2	4.5 ns	2.0 ns
3	5.5 ns	3.0 ns
4	6.5 ns	4.0 ns
5	7.5 ns	5.0 ns
6	8.5 ns	6.0 ns
7	9.5 ns	7.0 ns
8	10.5 ns	8.0 ns
9	11.5 ns	9.0 ns
10	12.5 ns	10.0 ns
11-15	クロック ディスエーブル	クロック ディスエーブル

戻り値: 無効。

副作用: なし

uint8 Clock_GetPhaseRegister(void)

説明: アナログ クロックの位相遅延値を取得します。この関数はアナログ クロックでのみ利用できます。

パラメータ: 無効。

戻り値: アナログ クロックの位相 (ナノ秒)。詳細は Clock_SetPhaseRegister() を参照してください。

副作用: なし

ファームウェア ソースコードの例

PSoC Creator は、「Find Example Project」 (プロジェクト例を検索) ダイアログに数多くのプロジェクト例を提供しており、そこには回路図およびコード例が含まれています。コンポーネント固有の例を見るには、「Component Catalog (コンポーネントカタログ) 」または回路図に置いたコンポーネントインスタンスからダイアログを開きます。一般例については、「Start Page (スタートページ) 」または **「File (ファイル)」** メニューからダイアログを開きます。必要に応じてダイアログにある **「Filter Options (フィルタのオプション) 」**を使用し、選択できるプロジェクトのリストを絞り込みます。

詳しくは、PSoC Creator ヘルプの「Find Example Project (プロジェクト例を検索)」を参照してください。

既知の問題

PSoC 5 では、cy_clock v1.50 コンポーネントは Clock_SetDividerRegister() 関数と一緒に使用しないでください。この関数をこのシリコンに使用された場合、既知の問題があります。この関数が必要な場合は、コンポーネントを最新のコンポーネントバージョンに更新してください。

コンポーネントの変更

ここでは、前のバージョンからコンポーネントに加えられた主な変更を示します。

バージョン	変更の説明	変更の理由 / 影響
1.50.b	データシートに既知の問題セクションを追加した。	
1.50.a	シリコン サポートの欠如について、データシートの Clock_StopBlock() にメモを追加しました	
	データシートのマイナーな編集と更新	
1.50	Clock_StopBlock() API を追加しました	この関数はクロックを停止し、ディスエーブルになるまで待ちます。これは設定を変更し、クロックをリスタートする際に問題を防ぐために必要です。
	Clock_GetPhaseRegister() API (アナログのみ) を追加しました	ファームウェアが現在のフェーズ バリューを読むことができます。
	Clock_SetPhaseValue() API (アナログのみ) を追加しました	このマクロは Clock_SetPhaseRegister() をラップし、自動的にフェーズ値に1を加算し、より直感的なインターフェースを提供します。
	Clock_SetPhase() を Clock_SetPhaseRegister() (アナログのみ) に名前を変更	他の名前と一貫性を保つためです。互換性のために、SetPhase がマクロとして提供され、Clock_SetPhaseRegister() と同じ効果があります。
	Clock_GetSourceRegister() API を追加しました	ファームウェアが現在のクロック ソースを読むことができます。
	Clock_SetSource() を Clock_SetPhaseRegister() に名前を変更しました	他の名前と一貫性を保つためです。互換性のために、SetSource がマクロとして提供され、Clock_SetSourceRegister() と同じ効果があります。
	Clock_GetModeRegister() API を追加しました	ファームウェアが現在のモード フラグを読むことができます。

バージョン	変更の説明	変更の理由 / 影響
	Clock_SetModeRegister() API を追加しました	この関数は Clock_SetMode() に置き換わります。互換性のために、SetMode がマクロとして提供され、Clock_SetModeRegister() と同じ効果があります。Clock_SetModeRegister() はモードフラグを0から1に変更するだけです。これは、SYNC のように、他のモード ビットを意図することなくクリアしないようにします。
	Clock_ClearModeRegister() API を追加しました	この関数は Clock_SetModeRegister() と似ていますが、モードフラグを1から0に変更するだけです。
	Clock_GetDividerRegister() API を追加しました	ファームウェアが現在のディバイダの値を読むことができます。
	Clock_SetDividerRegister() API を追加しました	Clock_SetDivider() API は無条件にクロック ディバイダをリセットします。Clock_SetDividerRegister() はファームウェアの作者がディバイダをリセットするかどうか制御できるようにします。
	Clock_SetDividerValue() API を追加しました	このマクロは Clock_SetDividerRegister() をラップし、自動的にディバイダから1を差し引き、より直感的なインターフェースを提供します。
	Clock_SetDividerRegister() に SSS を設定します。	1で分周する場合 (分周値が0)、SSS ビットは分周器をバイパスするよう設定しなければなりません。Clock_SetDividerRegister() 関数は自動的に SSS を設定/クリアし、必要に応じて一時的にクロックをディスエーブルにします。
	変更されたレジスタの定義	コンポーネントのコーディングと一致するためにアップデートしました
	修正された Clock_SetDivider() API ドキュメント	Clock_SetDivider() API ドキュメントには、clkDivider パラメータは分周値 + 1 であるべきことが示されています。しかし、分周値は -1 であったはずです。ドキュメントには、0 が clkDivider に対して無効な値であることが誤って示されています。
	「Synch with Bus」を「Sync with Master」に変更し、構成ダイアログ上のツールチップに関連付けた。	デバイスの仕組みに一致するようアップデートしました。これは単なる外見上の変化です。
	アナログ クロックからのデジタル ドメイン出力をイネーブルにするためにパラメータを追加しました。	以前はコンポーネント上で示されていなかった、ハードウェア内のアナログ クロックから信号を利用することができます。

バージョン	変更の説明	変更の理由 / 影響
	`=ReentrantKeil($INSTANCE_NAME . "_...") 」を次の関数に追加しました。 void Clock_Start() void Clock_Stop() void Clock_StopBlock() void Clock_StandbyPower() void Clock_SetDividerRegister() uint16 Clock_GetDividerRegister() void Clock_SetModeRegister() void Clock_ClearModeRegister() uint8 Clock_GetModeRegister() void Clock_SetSourceRegister() uint8 Clock_GetSourceRegister() void Clock_SetPhaseRegister() uint8 Clock_GetPhaseRegister()	ユーザが必要ならこれらの API を再び使用できるようにするため。
1.0a	Move CYCLK_ constants to *cydevice.h*/*cydevice_trm.h*.	モードとソースの CYCLK_ constants は選択されたデバイスのレジスタマップから生成されるようになりました。これにより、クロック コンポーネントはデバイス特有のレジスタ値と独立します。*cydevice.h* ファイルは既にクロックヘッダーに含まれているので、ユーザはコード変更をする必要がありません。
	データシートにCYCLK_ constantsの説明を追加した。	Clock_SetMode() および Clock_SetSource() API のパラーメータの説明は、各値の説明が含まれるようになりました。

Copyright © 2005-2012 Cypress Semiconductor Corporation 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporationは、サイプレス製品に組み込まれた回路以外のいかなる回路を使用することに対しても一切の責任を負いません。特許又はその他の権限下で、ライセンスを譲渡又は暗示することもありません。サイプレス製品は、サイプレスとの書面による合意に基づくものでない限り、医療、生命維持、救命、重要な管理、又は安全の用途のために仕様することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことを合理的に予想される、生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

PSoC Designer™及びProgrammable System-on-Chip™は、Cypress Semiconductor Corp.の商標、PSoC®は同社の登録商標です。本文書で言及するその他全ての商標又は登録商標は各社の所有物です。

全てのソースコード(ソフトウェア及び/又はファームウェア)はCypress Semiconductor Corporation (以下「サイプレス」)が所有し、全世界(米国及びその他の国)の特許権保護、米国の著作権法並びに国際協定の条項により保護され、かつそれらに従います。サイプレスが本書面によるライセンシーに付与するライセンスは、個人的、非独占的かつ譲渡不能のライセンスであって、適用される契約で指定されたサイプレスの集積回路と併用されるライセンシーの製品のみをサポートするカスタムソフトウェア及び/又はカスタムファームウェアを作成する目的に限って、サイプレスのソースコードの派生著作物を複製、使用、変更、そして作成するためのライセンス、並びにサイプレスのソースコード及び派生著作物をコンパイルするためのライセンスです。上記で指定された場合を除き、サイプレスの書面による明示的な許可なくして本ソースコードを複製、変更、変換、コンパイル、又は表示することは全て禁止されます。

免責条項:サイプレスは、明示的又は黙示的を問わず、本資料に関するいかなる種類の保証も行いません。これには、商品性又は特定目的への適合性の黙示的な保証が含まれますが、これに限定されません。サイプレスは、本文書に記載される資料に対して今後予告なく変更を加える権利を留保します。サイプレスは、本文書に記載されるいかなる製品又は回路を適用又は使用したことによって生ずるいかなる責任も負いません。サイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

ソフトウェアの使用は、適用されるサイプレスソフトウェアライセンス契約によって制限され、かつ制約される場合があります。