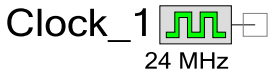
クロック
1.60

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
- クロックの周波数の許容範囲を設定可能

概要説明

クロック コンポーネントには、ローカルクロックの作成、システムおよびデザイン全体でのクロックのデザインという 2 つの主要な機能があります。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。詳細は PSoC Creator ヘルプの「クロック エディタ」セクションを参照してください。

クロックは複数の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

周波数を指定した場合、PSoC Creator は自動的に最も正確な周波数が生成されるディバイダを選択します。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。

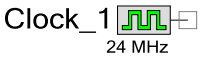
- デジタル - 波形の色はデジタルワイヤーと同じ色で、黒色のアウトラインがあります。
- アナログ - 波形の色はアナログワイヤーと同じ色で、黒色のアウトラインがあります。
- 不確定 - 波形の色は白で、アウトラインはありません

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。

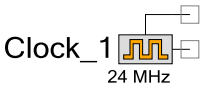
クロック - 出力

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 *

「Force clock to be Analog Clock (クロックをアナログ クロックに強制する)」
が選択されると、このオプションの出力は、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。**「構成」**ダイアログの**「詳細設定」**タブのオプションを利用して、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**「構成」**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。詳細は、このデータシートのアプリケーション プログラミング インタフェースセクションおよび PSoC Creator ヘルプの「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 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>

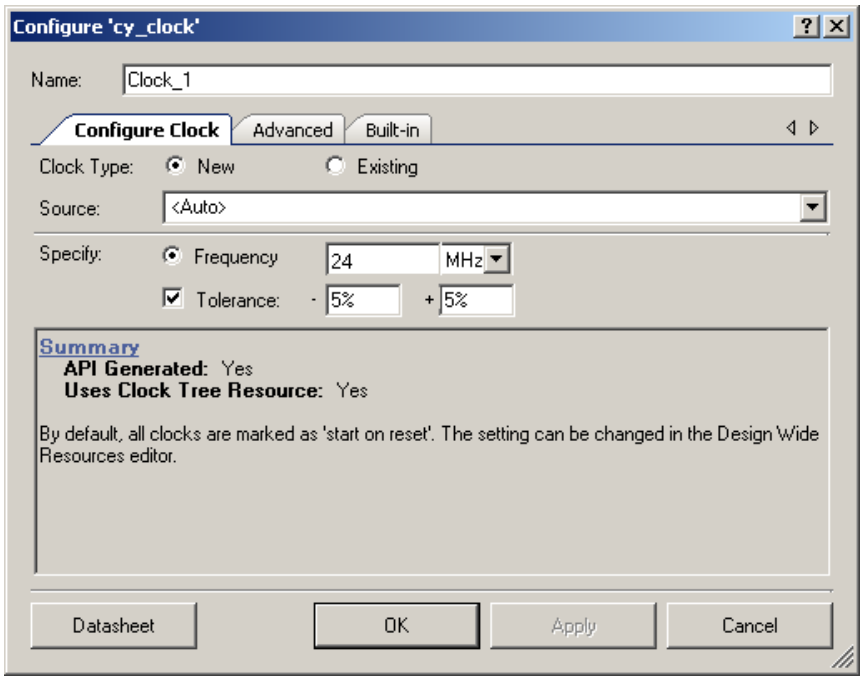


図 2.Clock Type (クロックのタイプ): New / Source: 特定のクロック

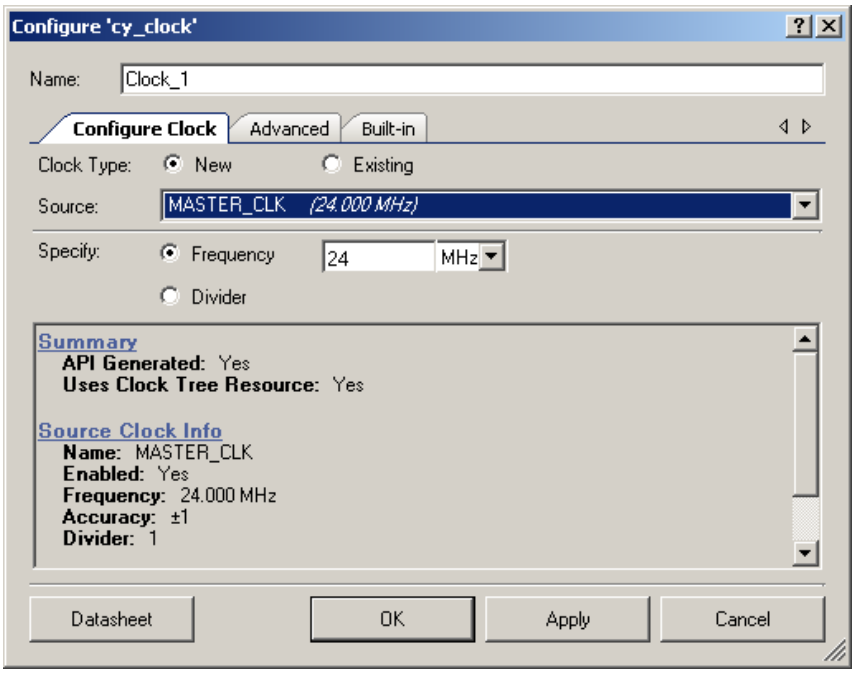
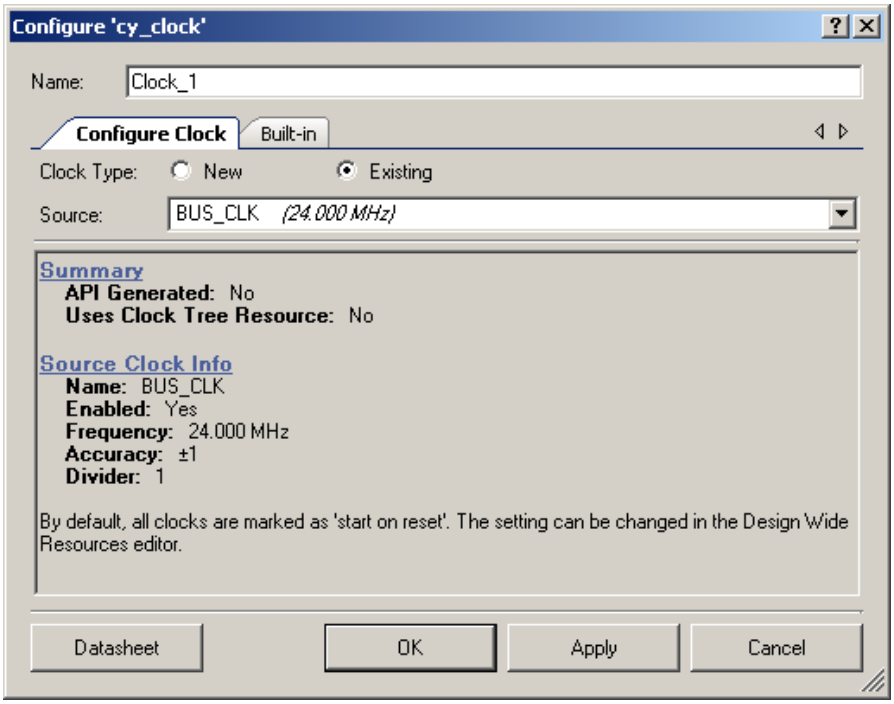


図 3.Clock Type (クロックのタイプ): 既存

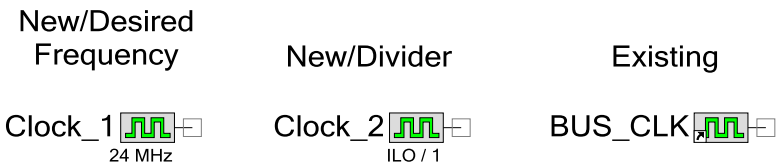


次のセクションでは、クロック コンポーネントのパラメータを説明します。

Clock Type (クロックのタイプ):

2 つのクロックタイプ:**「新規」**および**「既存」**があります。新しいクロックに対しては、使用するクロック**「ソース」**を指定、または<Auto>を選択して、PSoC Creator に選択させることができます。<Auto>を選択した場合、特定の**「周波数」**およびオプションの**「許容範囲」**も入力できます。**「ソース」**を指定した場合、**「周波数」**を指定するか、**「ディバイダ」**を選択することができます。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」として構成されたクロックコンポーネントは、デバイスのクロックリソースを消費し、API が作成されます。システムまたはデザイン全体のクロックに対して**「既存」**として構成されたクロックコンポーネントは、デバイス上で物理リソースを消費せず、API は作成されません。代わりに、選択したシステムまたはデザイン全体のクロックを使用します。

Source (ソース)

自動的に利用可能なソースクロックをPSoC Creator で検索し、分周した場合に、最も正確な周波数にするためには<Auto> (デフォルト) を選択します。<Auto>をソースに持つクロックは、希望する周波数のみ入力できます。オプションとして、許容範囲を提供することもできます。

提供されたリストからシステムまたはデザイン全体のクロックを選択し、PSoC Creator がそのクロックをソースとして使用するよう強制します。

周波数

希望する周波数およびユニット (既定値 = **24 MHz**) を入力します。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

<Auto>をクロック ソースとして選択した場合、クロックの希望許容値を入力することができます(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で指定します (**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。) 希望する許容範囲がない場合、許容範囲の隣にあるチェックボックスを選択解除します。そのクロックに対して警告は発生しません。

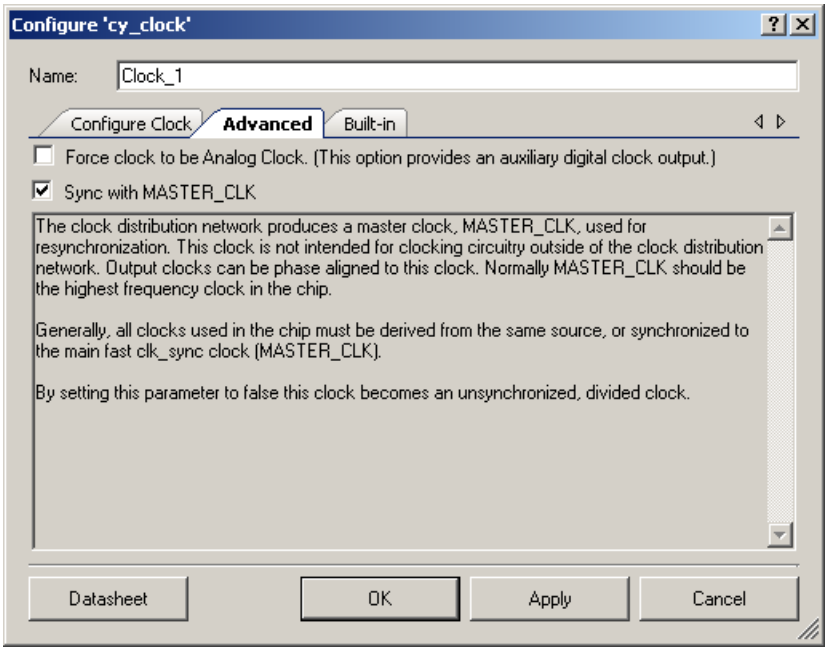
Divider (ディバイダ-分周器)

特定の**ソース**を選択した場合、**ディバイダ**の明示的な値を入力することができます。**ソース**が<Auto>のままの場合、**ディバイダ**オプションは利用できません (既定値)。

ディバイダオプションを選択した場合、**周波数**オプションが利用できません。

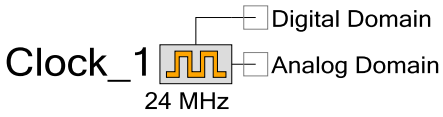
Advanced Tab（詳細設定タブ）

詳細設定タブには 2 つのパラメータがあります。



クロックをアナログ クロックに強制する

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 ドメインにあります。



MASTER_CLK で同期化する

選択した場合 (デフォルトは選択されていない) 、クロックは MASTER クロックと同期化されますが、それ以外の場合はクロックは同期化されません。

配置およびリソース

リソースの使用は構成および接続性によります。

- **既存**として構成されているクロック コンポーネントは、チップ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の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。

アナログ ブロック	デジタル ブロック					API メモリ(バイト)		ピン (外部入出力ごと)
	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	
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、与えられたユーザの回路図に最初に配置した制御レジスタインスタンス名として「Clock_1」を割り当てます。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 **クロック タイプ**で構成されたローカルクロックで、Configureダイアログで**既存**に設定されている場合、API は生成されません。

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。

機能	説明
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。

注意 PsoC 5 シリコンを使用するときソース「MASTER_CLK」およびディバイダ1を備えたクロックが必ず稼働しています。

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。

注意 PSoC 3 ES2 および PSoC 5 シリコンではClock_StopBlock() API はサポートされず、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有効になります。

パラメータ: uint16 clkDivider: ディバイダのレジスタ値 (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: ディバイダのレジスタ値 (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、クロック分周器がリスタートし、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有効になります。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有効になります。

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有効になります。

パラメータ: uint16 clkDivider: 分周値 (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 ディバイダのレジスタ値を取得します。

パラメータ: 無効

戻り値: クロックの値から 1 を引いたものを分周します。例えば、クロックが2で分周するよう設定されている場合、戻り値は1です。

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% デューティーサイクル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: マスタークロックへの出力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% デューティーサイクル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: マスタークロックへの出力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: 無効

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。モードビットの詳細は Clock_SetModeRegister() および Clock_ClearModeRegister() の説明を参照してください。

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。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% デューティーサイクル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: マスタークロックへの出力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: フェーズ遅延マスター 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: フェーズロック 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: フェーズ遅延マスター 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: フェーズロック 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。問題を避けるために、フェーズ遅延を変更する前に、位相遅延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 生産以降、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。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 生産以降、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。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 生産以降、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 クロックのフェーズ (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。

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
1.60	Updated Clock_SetDivider() および Clock_SetDividerRegister() API	PSoC 5 と適切に機能する固定 API
	「デジタルドメイン - 出力」の表現を変えました	
	データシートの Clock_Stop() にメモを追加しました	
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。

バージョン	変更の説明	変更の理由 / 影響
	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。
	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 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に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。
	Configure（構成) ダイアログにおいて、「Synch with Bus」(バスと同期化)を「Sync with Master」(マスターと同期化)および関連するツールチップに変更しました。	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。
1.0.a	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 の説明を追加しました。	Clock_SetMode() および Clock_SetSource() API のパラメータの説明は、各値の説明が含まれるようになりました。

© Cypress Semiconductor Corporation, 2012. 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 (サイプレスセミコンダクタ社) 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。かつ、サイプレスセミコンダクタコーポレーションは、特許またはその他の権利に基づくライセンスを譲渡することも、又は含意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、または安全の用途のために使用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ト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® は、サイプレス セミコンダクタ社の登録商標であり、PSoC Creator™ およびプログラマブル System-on-Chip™ は、サイプレスセミコンダクタ社の商標です。本書で言及するその他すべての商標または登録商標は、各社の所有物です。

全てのソースコード（ソフトウェアおよび / またはファームウェア）はサイプレスセミコンダクタコーポレーション（以下「サイプレス」）が所有し、全世界の特許権保護（米国およびその他の国）、米国の著作権法なら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り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および / また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コピー、使用、変更そして作成するためのライセンス、ならびにサイプレスのソースコードおよ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、また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、明示的または黙示的を問わず、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、商品性また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または回路を適用また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ト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。