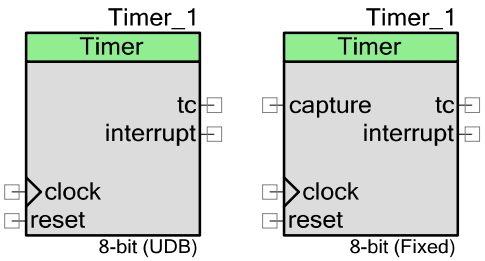
タイマ
2.20

特長



- 専用機能ブロック(Fixed-function: FF)実装と UDB(universal digital block)実装
- 8、16、24、32 ビットタイマ
- キャプチャ入力(オプション)
- 他のコンポーネントと同期をとるためのイネーブル、トリガ、リセット入力
- 連続またはワンショット実行モード

概要

タイマコンポーネントは時間間隔を測定する方法を提供します。基本的なタイマ関数を実装したり、キャプチャカウンタまたは割り込み/DMA 生成によるキャプチャなどの詳細な機能の提供が可能です。

このコンポーネントは FF ブロックまたは UDB を使って実装できます。UDB 実装は通常、FF 実装より多くの機能を持ちます。簡単な設計の場合は、FF を使って UDB リソースを他の目的にとっておくことを考慮してください。

次の表に、FF と UDB の主な違いを示します。FF と UDB 実装の間には多くの機能的な違いがあります。また FF 実装はデバイスによって違いがあります。様々な実装の詳細なタイミング波形については設定セクションをご覧ください。

機能	FF	UDB
ビット数	8、16	8、16、24、32
Run Mode (動作モード)	連続、ワンショット	連続、ワンショット、割り込みで停止するワンショット
カウントモード	ダウンのみ	ダウンのみ
イネーブル入力	有 (ハードウェアまたはソフトウェアイネーブル)	有 (ハードウェアまたはソフトウェアイネーブル)
キャプチャ入力	有	有
Capture mode (キャプチャモード)	立ち上がりエッジのみ	立ち上がりエッジ、立ち下がりエッジ、両エッジ、またはソフトウェア制御
キャプチャ FIFO	無 (キャプチャレジスタ 1 個)	有 (キャプチャレジスタ 4 個まで)

機能	FF	UDB
トリガ入力	無	有
Trigger Mode (トリガモード)	無	立ち上がりエッジ、立ち下がりエッジ、両エッジ、またはソフトウェア制御
リセット入力	有	有
ターミナルカウント出力	有	有
割り込み出力	有 (PSoC 3 のみ)	有
割り込み条件	TC、キャプチャ	TC、キャプチャ、FIFO フル
キャプチャ出力	無	有
周期レジスタ	有	有
周期リロード	有 (リセットまたは TC 時に常にリロード)	有 (リセットまたは TC 時に常にリロード)
クロック入力	クロックシステム内のデジタル クロックに限定	任意の信号

タイマの用途

タイマのデフォルト用途は周期イベントまたは割り込み信号の生成です。しかし、その他の用途もあります:

- クロックをクロック入力に入力し、ターミナルカウント出力を分周クロック出力として使用するクロック分周器をつくること。
- クロックをクロック入力に入力し、テスト信号をイネーブル入力またはキャプチャ入力に入力して、ハードウェアイベント間の時間差を測定すること。

注 イベントの計数に焦点を置く場合はカウンタコンポーネントが適しています。PWM コンポーネントは、センターアライメント、出力キル、デッドバンド出力などの制御機能を持つ複数の比較出力を必要とする状況に適しています。

標準的な使用方法では、タイマはイベント間のクロック数を記録します。この例としては、回転速度計センサによって発生するような 2 つの立ち上がりエッジ間のクロック数を測定します。より複雑な使用法としては、PWM 入力の周期とデューティサイクルの測定があります。PWM 測定では、タイマコンポーネントを立ち上がりエッジで開始させ、次の立ち下がりエッジをキャプチャし、次の立ち上がりエッジでキャプチャおよび停止するように設定します。最後のキャプチャによる割り込みにより、キャプチャされたすべての値が FIFO に用意されていることが CPU に伝達されます。

入出力の接続

このセクションでは、タイマのさまざまな入出力接続について説明します。一部の I/O のシンボルは、その I/O の説明に示される条件において、表示されないことがあります。

注 特記がない限り、すべての信号はアクティブハイです。

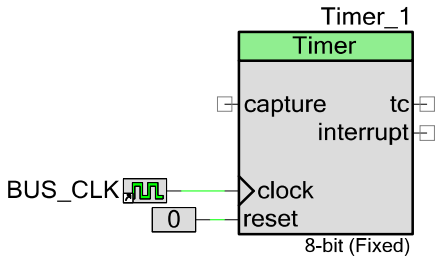
入力	非表示の場合	説明
clock	N	clock 入力は、Timer コンポーネントの動作周波数を決めます。つまり、Timer コンポーネントが有効になっている時には、clock 入力の立ち上がりエッジで、タイマ周期カウントがデクリメントされます。
reset	N	この入力は同期リセットです。この入力は、少なくともクロックの立ち上がりエッジひとつ分が必要で、カウントとキャプチャカウンタをリセットするように実装されています。reset 入力は、周期カウンタを周期値にリセットします。またキャプチャカウンタもリセットされます。
enable	Y	この入力はタイマハードウェアイネーブルです。クロックの各立ち上がりエッジで周期カウンタがデクリメントする動作をこの入力で有効にします。この入力がローレベルの場合、出力はアクティブのままですが、タイマコンポーネントの状態は変わりません。この入力は、**Enable Mode** パラメーターが **Hardware Only** または **Software and Hardware**に設定されているときに表示されます。
capture	Y	キャプチャ入力は、現在のカウント値をキャプチャ レジスタまたは FIFO に取り込みます。この入力は、**Capture Mode** パラメーターが **None**以外のモードに設定されているときに表示されます。キャプチャは **Capture Mode** 設定に従い、この入力に適用された立ち上がりエッジ、立ち下がりエッジ、または両方のエッジで発生します。capture 入力は clock 入力のタイミングでサンプルされます。タイマが無効になっている場合、カウントはキャプチャされません。キャプチャ入力に外部接続がされずフローティング状態の場合もあります。キャプチャラインに何も接続していない場合は、コンポーネントが定数ロジック 0 を割り当てます。
trigger	Y	トリガ入力は設定可能なハードウェアイベントに基づいてタイマのカウント開始または停止を有効にします。この入力は、**Trigger Mode** パラメーターが **None**以外のモードに設定されているときに表示されます。これにより、適切なエッジが検知されるまで、タイマのカウントが遅れることになります。トリガエッジは、キャプチャもされなければ、割り込みも発生させません。

出力	非表示の場合	説明
tc	N	Terminal Countは、カウントが0に等しいことを示す同期出力です。出力は、タイマのクロック入力に同期されます。この出力の正確なタイミングは、デバイスと、UDB または FF 実装が使用されているかによります。
interrupt	N	interrupt 出力は、ハードウェアで設定された割り込みソースによって駆動されます。全ソースの論理和をとって、最終的な出力信号が作成されます。割り込みのソースは Terminal Count、Capture、FIFO full。 割り込みがトリガされると、ステータスレジスタが読まれるまで割り込み出力はアサートされたままです。 割り込み接続はPSoC 5 での FF 実装には対応していません。この機能が必要な場合は、割り込みコンポーネントを tc 信号に接続するか、UDB 実装を使用することが可能です。

出力	非表示の場合	説明
capture_out	Y	キャプチャ出力は、ハードウェアキャプチャがトリガされたことを示します。この信号は、UDB実装の時に限り利用できます。出力は、タイマのクロック入力に同期されます。

回路図マクロ情報

Component Catalog にある初期設定のタイマは、初期設定のタイマコンポーネントを使用した回路図マクロです。このマクロには、バスクロックとロジックローコンポーネントが接続されます。



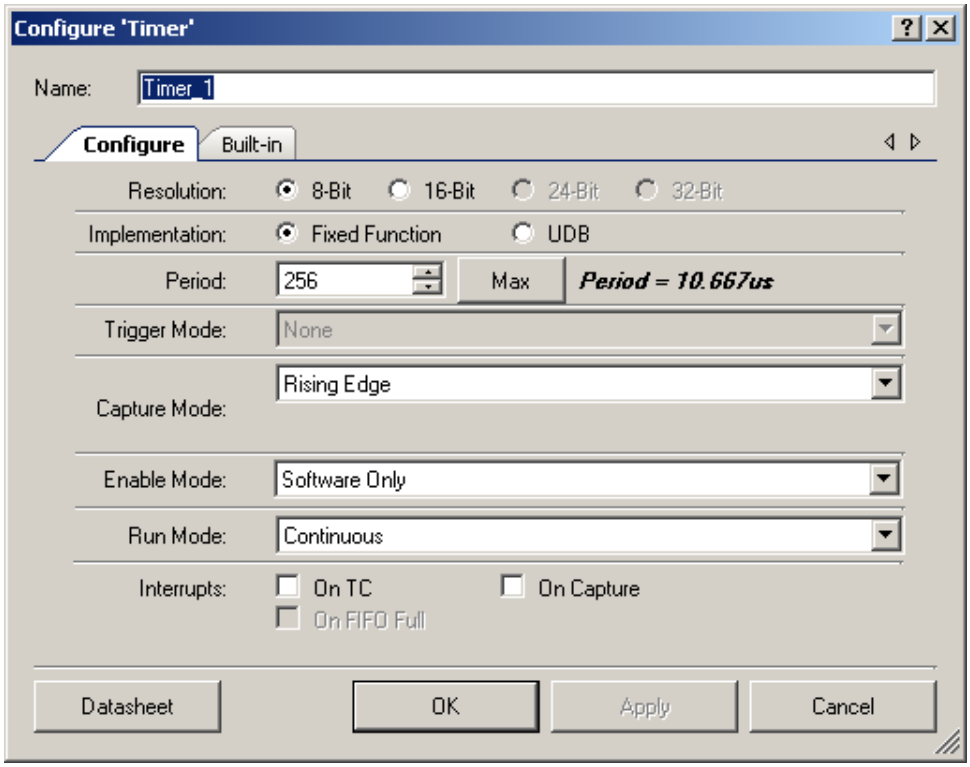
コンポーネント パラメータ

タイマコンポーネントを回路図上にドラッグし、ダブルクリックして **Configure** ダイアログを開きます。

ハードウェアとソフトウェアの構成オプション

ハードウェア構成オプションは、ハードウェア中でコンポーネントが合成・配置される方法を変えます。これらのオプションを変更する場合、ハードウェアをリビルドしなければなりません。ソフトウェア構成オプションは、合成や配置に影響を及ぼしません。ビルドの前にこれらのパラメータを設定することは、初期値を設定することになります。パラメーターは提供されている API を用いていつでも修正できます。後述のパラメータの殆どは、ハードウェアオプションです。

Configure タブ



Resolution

Resolution パラメータは、タイマのビット幅を定義します。この値は最大カウント 255、65535、16777215、4294967295 に対して、それぞれ 8、16、24、32 に設定できます。FF 実装では、Resolution は 8 または 16 に制限されます。

Implementation

Implementation パラメータを使うと、タイマを専用機能ブロックで実装するか、または UDB で実装するかを選択できます。FF を選択すると、UDB 関数は無効になります。

Period (ソフトウェアオプション)

Period パラメーターはカウンタの周期を定義します。タイマコンポーネントの最大カウント値(またはロールオーバーポイント) は **Period** から 1 を減算した値になります。**Period** から 1 を減算した値は周期レジスタに読み込まれる初期設定値です。ソフトウェアは Timer_WritePeriod() API でこのレジスタをいつでも変更することが可能です。API を使って同等の結果を得るためには、カスタマイザからの **Period** 値から、1 を減算した値を、この関数の独立変数として使う必要があります。

この値の範囲は、**Resolution** パラメータで定義します。8、16、24、32 ビットの **Resolution** では、**Period** は: それぞれ 2^8、2^16、2^24、2^32 つまり 256、65536、16777216、4294967296 です。

Trigger Mode (ソフトウェアオプション)

Trigger Mode パラメーターはトリガ入力の実装を設定します。このパラメーターは **Implementation** が **UDB** に設定されているときのみ利用できます。

Trigger Mode は以下のいずれかの値に設定できます:

- **None** (default) – トリガは実装されず、トリガ入力ピンは表示されません。
- **Rising Edge** – トリガ入力を有効にし、トリガ入力の最初の立ち上がりエッジでカウントします。
- **Falling Edge** – トリガ入力を有効にし、トリガ入力の最初の立下りエッジでカウントします。
- **Either Edge** – トリガ入力を有効にし、トリガ入力の最初のエッジ(立ち上がりまたは立下り)でカウントします。
- **Software Controlled –** トリガモードは動作中に Timer_SetTriggerMode() API 呼び出しを使用して、上記 4 つのトリガモードのいずれかに設定可能です。デフォルトは、この API を使って他の値を設定するまでは **None** です。

Capture Mode (ソフトウェアオプション)

キャプチャモードセクションには 3 つのパラメーターがあります: **Capture Mode**, **Enable Capture Counter**、および **Capture Count** です。

Capture Mode

Capture Mode パラメータは、キャプチャのタイミングを設定します。キャプチャ入力はクロック入力の立ち上がりエッジでサンプルされます。このモードは以下の値のいずれかに設定可能です(専用機能ブロック実装では、**None** と **Rising Edge** のみが利用可能です。):

- **None** – キャプチャは実装されず、キャプチャ入力ピンは隠されます。
- **Rising Edge** – クロック入力に対するキャプチャ入力の立ち上がりエッジでカウントをキャプチャします。
- **Falling Edge**– クロック入力に対するキャプチャ入力の立ち下がりエッジでカウントをキャプチャします。
- **Either Edge**– クロック入力に対するキャプチャ入力の両エッジでカウントをキャプチャします。
- **Software Controlled** — キャプチャモードは動作中に Timer_SetCaptureMode() API 呼び出しを使用して、上記 4 つのキャプチャモードのいずれかに設定可能です。デフォルトは、この API を使って他の値を設定するまでは **None** です。

Enable Capture Counter (ソフトウェアオプション)

Enable Capture Counter パラメーターは、カウンタが実際にキャプチャされる前に、いくつのキャプチャイベントが発生するかを定義します。例えば、3 イベント毎にキャプチャが必要な場合は、キャプチャカウンタ値を 3 に設定します。このパラメーターは、UDB 実装でのみ利用可能です。

Capture Count (ソフトウェアオプション)

Capture Count パラメーターは、カウンタが実際にキャプチャされる前に発生するキャプチャイベントの初期設定値を設定します。このパラメータには、2～127 の値を設定できます。キャプチャカウント値は動作中に API 関数 Timer_SetCaptureCount を呼び出すことで修正可能です。このパラメーターは、UDB 実装でのみ利用可能です。

Enable Mode

Enable Mode パラメータは、タイマのイネーブル実装を設定します。イネーブル入力はクロック入力の立ち上がりエッジでサンプルされます。このモードは、以下のいずれかの値に設定できます。

- **Software Only** – タイマは、コントロールレジスタのイネーブル ビットのみに基づいて有効になります。
- **Hardware Only** – タイマは、イネーブル 入力のみに基づいて有効になります。(UDB のみ)
- **Software and Hardware** – タイマは、ハードウェアとソフトウェアのイネーブルが共に「真」のときに有効になります。

Run Mode

Run Mode パラメータは、タイマコンポーネントの動作を連続動作とワンショットモード動作のいずれかから設定できます:

- **Continuous** – タイマが有効であれば、連続動作します。
- **One Shot** – タイマはカウントを開始し、 0 に達すると停止します。リセット後、別の 1 サイクルを開始します。UDB タイマの場合は停止時に **Period** をカウントレジスタにリロードし、専用機能ブロックタイマの場合はカウントレジスタはカウント終了時にも残ります。
- **One Shot (Halt on Interrupt)** – タイマはカウントを開始し、0 に達するか割り込み発生時に停止します。リセット後、別の 1 サイクルを開始します。UDB タイマの場合は停止時に **Period** をカウントレジスタにリロードし、専用機能ブロックタイマの場合はカウントレジスタはカウント終了時にも残ります。

  注 ワンショットモードを設定する時にタイマが予期せず動作を始めないようにするため、**Trigger Mode** を使用して動作開始時期を制御するか、ソフトウェアによるイネーブル(**Software Only** または **Software and Hardware**) を使用すべきです。

Interrupt (ソフトウェアオプション)

Interrupt パラメータを使うと、初期割り込みソースを設定できます。割り込みは以下のうち選択したイベントが発生したときに発生します。このモードはいつでもソフトウェアで再設定できます。このパラメータは単に初期設定を定義するだけです。

- **On TC** – このパラメータは常にアクティブです。初期設定は無効です。
- **On Capture (1-4) –** 所定のキャプチャ数で割りこみを可能にします。初期設定は無効です。
- **On FIFO Full –** キャプチャ FIFO がフルの場合に割り込みを可能にします。初期設定は無効です。

クロックの選択

PSoC 3 または PSoC 5 クロックシステムの詳細は、クロック コンポーネントのデータシートと該当するデバイスのデータシートを参照してください。

専用機能ブロックコンポーネント

デバイスの FF ブロックを使用するように設定されている場合、タイマコンポーネントは次の制限を受けます。

- クロック入力はクロックシステムからのデジタルクロックでなければなりません。
- クロックの周波数がバスクロックと等しい場合、クロックは実際にバスクロックでなければなりません。

  該当するクロックコンポーネントの **Configure** ダイアログを開いて、**Clock Type** パラメーターを **Existing** として設定し、**Source** パラメーターを **BUS_CLK** として設定します。この周波数のクロックは、マスタクロックや IMO など、他のソースから分周できません。

UDB ベースのコンポーネント

どのソースからのデジタル信号もクロック入力に接続できます。この信号の周波数は、このデータシートの DC/ AC 電気的特性(UDB 実装) セクションで定義している周波数範囲に制限されます。

配置

PSoC Creator は、**Implementation** パラメータに基づいてタイマ コンポーネントをデバイス内に配置します。**Fixed Function** に設定すると、このコンポーネントは使用可能な任意の FF カウンタ/タイマブロックに配置されます。**UDB** に設定すると、このコンポーネントは最適な設定で UDB アレイに設置されます。

リソース

Resolution	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	
8ビット UDB タイマ [1]	1	6	1	1	0	257	5	-
8-bit FF タイマ [2]	0	0	0	0	0	234	2	-
16 ビット UDB タイマ[1]	2	6	1	1	0	295	6	-
16 ビット FF タイマ [2]	0	0	0	0	0	248	2	
24 ビット UDB タイマ [1]	3	6	1	1	0	287	8	-
32 ビット UDB タイマ [1]	4	6	1	1	0	287	8	-
8 ビットUDB タイマ ワンショット [3]	1	8	1	1	0	257	5	-
16 ビット UDB タイマ ワンショット [3]	2	8	1	1	0	295	6	-

[1] UDB タイマは Software Only Enable mode、Rising Edge Trigger mode、Continuous Run mode、Interrupt on TC with no Capture mode に設定されます。

[2] FF タイマは Software Only Enable mode、Rising Edge Trigger mode、Continuous Run mode、Interrupt on TC に設定されます。

[3] UDB タイマは Software Only Enable mode、Rising Edge Trigger mode、One Shot mode Interrupt on TC with no Capture mode に設定されます。

アプリケーション プログラミング インターフェース

アプリケーション プログラミング インタフェース (API) 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。以後のセクションで、各関数について詳しく説明します。

初期設定では、PSoC Creator は、ユーザの回路図に最初に配置されたコンポーネントのインスタンス名として "Timer_1"を割り当てます。コンポーネントの名称は、識別子の文法ルールに従って固有の名称に変更できます。インスタンス名は、すべてのグローバル関数名、変数名、定数名の接頭辞になります。便宜上、次の表では "Timer"というインスタンス名を使用します。

関数	機能
Timer_Start()	initVar 変数を設定し、Timer_Init() 関数を呼び出して、Enable 関数を呼び出します。
Timer_Stop()	タイマを無効にします。
Timer_SetInterruptMode()	割り込み出力のソースを有効または無効にします。
Timer_ReadStatusRegister()	ステータスレジスタの現在の状態を返します。
Timer_ReadControlRegister()	コントロールレジスタの現在の状態を返します。
Timer_WriteControlRegister()	コントロールレジスタのビットフィールドを設定します。
Timer_WriteCounter()	新しい値を直接カウンタレジスタに書き込みます。(UDBのみ)
Timer_ReadCounter()	キャプチャを強制し、キャプチャ値を返します。
Timer_WritePeriod()	周期レジスタに書き込みます。
Timer_ReadPeriod()	周期レジスタを読み取ります。
Timer_ReadCapture()	キャプチャレジスタの内容または FIFO の出力を返します。
Timer_SetCaptureMode()	キャプチャが発生する、ハードウェアまたはソフトウェア条件を設定します。
Timer_SetCaptureCount()	カウンタレジスタをFIFOにキャプチャする前に、カウントするキャプチャイベント数を設定します。
Timer_ReadCaptureCount()	キャプチャイベント数の現在の設定を読み出します。
Timer_SoftwareCapture()	カウント値のキャプチャFIFOへのキャプチャを強制します。
Timer_SetTriggerMode()	トリガが発生するハードウェアまたはソフトウェア条件を設定します。
Timer_EnableTrigger()	タイマのトリガモードを有効にします。
Timer_DisableTrigger()	タイマのトリガモードを無効にします。
Timer_SetInterruptCount()	割り込みがトリガされる前にカウントするキャプチャ数を設定します。
Timer_ClearFIFO()	キャプチャ FIFO をクリアします。

関数	機能
Timer_Sleep()	タイマを停止し、現在の設定を保存します。
Timer_Wakeup()	タイマの設定を復元し、タイマを再度有効にします。
Timer_Init()	Configureダイアログでの設定に従ってタイマを初期化または復元します。
Timer_Enable()	タイマを有効にします。
Timer_SaveConfig()	タイマの現在の設定を保存します。
Timer_RestoreConfig()	タイマの設定を復元します。

グローバル変数

変数	説明
Timer_initVar	タイマが初期化済みかを示します。この変数は、0に初期化され、Timer_Start()が最初に呼び出されたときに1にセットされます。この変数を使うと、コンポーネントは Timer_Start() ルーチンを一度呼び出した後、再初期化することなく再起動できます。 コンポーネントの再初期化が必要な場合には、Timer_Start()やTimer_Enable()関数の前に、Timer_Init()関数を呼び出します。

void Timer_Start(void)

機能: これは、コンポーネントの動作を開始する際に推奨される方法です。Timer_Start() は initVar 変数を設定し、Timer_Init() 関数をび出してから Timer_Enable() 関数を呼び出します。

パラメータ: なし

返り値: なし

注意事項: 変数 initVar がすでに設定されている場合、この関数は単にTimer_Enable()関数を呼び出します。

void Timer_Stop(void)

機能: 専用機能ブロック実装では、この関数はタイマを無効化し、電源を切ります。UDB 実装では、タイマはソフトウェアイネーブルモードでしか無効になりません。

パラメータ: なし

返り値: なし

注意事項: 専用機能ブロックタイマはこの関数で電源が切られるため、TC 出力はローレベルになります。

void Timer_SetInterruptMode(uint8 interruptMode)

機能: 割り込み出力のソースを有効または無効にします。

パラメータ: uint8: 割り込みソース。ビット定義については、このデータシートの モードレジスタ セクションをご覧ください。

返り値: なし

注意事項: ビット位置は FF と UDB で異なります。Mask #defines はこの差異をカプセル化するために提供されています。

uint8 Timer_ReadStatusRegister(void)

機能: ステータス レジスタの現在の状態を返します。

パラメータ: なし

返り値: uint8: 現在のステータスレジスタ値
ビット定義については、このデータシートの ステータスレジスタ セクションを参照ください。

注意事項: これらのビットのいくつかは、ステータスレジスタが読み取られるとクリアされます。Clear-on-read ビットは、このデータシートの ステータスレジスタ セクションで定義しています。

uint8 Timer_ReadControlRegister(void)

機能: コントロールレジスタの現在の状態を返します。この API はコントロールレジスタを必要としない特別なケースでは利用できません(UDB 実装、イネーブルモードがハードウェアのみ、キャプチャモードがソフトウェア制御ではない、トリガモードがソフトウェア制御ではない場合など)。

パラメータ: なし

返り値: uint8: コントロールレジスタ ビットフィールド
ビット定義については、このデータシートの コントロールレジスタ セクションを参照ください。

注意事項: なし

void Timer_WriteControlRegister(uint8 control)

機能: コントロールレジスタのビットフィールドを設定します。この API はコントロールレジスタを必要としない特別なケースでは利用できません(UDB 実装、イネーブルモードがハードウェアのみ、キャプチャモードがソフトウェア制御ではない、トリガモードがソフトウェア制御ではない場合など)。

パラメータ: uint8: コントロールレジスタ ビットフィールド
ビット定義については、このデータシートの コントロールレジスタ セクションを参照ください。

返り値: なし

注意事項: なし

void Timer_WriteCounter(uint8/16/32 counter)

機能: 新しい値を直接カウンタレジスタに書き込みます。この関数は UDB 実装でのみ使用可能です。

パラメータ: uint8/16/32: 新しいカウンタ値。24 ビットタイマでは、このパラメータは uint32 です。

返り値: なし

注意事項: カウントを上書きします。これは、ターミナルカウント出力、または周期幅で望ましくない動作を引き起こすことがあります。これはアトミック書き込みではないので、関数に割り込みが起きる可能性があります。この関数を呼び出す前にタイマを無効にしてください。

uint8/16/32 Timer_ReadCounter(void)

機能: キャプチャを強制し、キャプチャ値を返します。

パラメータ: なし

返り値: uint8/16/32: 現在のカウント。24 ビットタイマでは、返り値の型は uint32 です。

注意事項: キャプチャレジスタの内容または FIFO の出力を返します (UDB のみ)。

void Timer_WritePeriod(uint8/16/32 period)

機能: 周期レジスタに書き込みます。

パラメータ: uint8/16/32: 新しい周期値。24 ビット タイマでは、このパラメータは uint32 です。

返り値: なし

注意事項: タイマの周期は、カウンタが周期レジスタからリロードされるまで変更されません。

uint8/16/32 Timer_ReadPeriod(void)

機能: 周期レジスタを読み取ります。

パラメータ: なし

返り値: uint8/16/32: 現在の周期値。24 ビットタイマでは、返り値の型は uint32 です。

注意事項: なし

uint8/16/32 Timer_ReadCapture(void)

機能: キャプチャレジスタの内容または FIFO の出力を返します (UDB)。

パラメータ: なし

返り値: uint8/16/32: 現在のキャプチャ値 24 ビットタイマでは、返り値の型は uint32 です。

注意事項: UDB 実装では、この値は FIFO から削除されます。

void Timer_SetCaptureMode(uint8 captureMode)

機能: キャプチャ モードを設定します。この機能は、UDB 実装および **Capture Mode** パラメーターが **Software Controlled** に設定されている場合のみ利用可能です。

パラメータ: uint8: 列挙型キャプチャ モード。コントロールレジスタセクションも参照してください:

Timer__B_TIMER__CM_NONE
Timer__B_TIMER__CM_RISINGEDGE
Timer__B_TIMER__CM_FALLINGEDGE
Timer__B_TIMER__CM_EITHEREDGE
Timer__B_TIMER__CM_SOFTWARE

返り値: なし

注意事項: なし

void Timer_SetCaptureCount(uint8 captureCount)

機能: キャプチャが実行される前に、検出するキャプチャイベント数を設定します。この関数は、UDB 実装で、Configureダイアログで**Enable Capture Counter**パラメーターが選択されている場合のみ利用可能です。

パラメータ: uint8 captureCount: キャプチャFIFO にカウントをキャプチャする前に、カウントしたいキャプチャイベントの数。2 ～127 までの値が有効です。

返り値: なし

注意事項: なし

uint8 Timer_ReadCaptureCount(void)

機能: Timer_SetCaptureCount() 関数で設定される形式で、captureCount パラメーターの現在の設定値を読み出します。この関数は、UDB 実装で Configureダイアログで**Enable Capture Counter**パラメーターが選択されている場合のみ利用可能です。

パラメータ: なし

返り値: uint8: 現在のキャプチャ数

注意事項: なし

void Timer_SoftwareCapture(void)

機能: 強制的にソフトウェアで現在のカウントをFIFOへキャプチャします。この関数は UDB 実装でのみ使用可能です。

パラメータ: なし

返り値: なし:

注意事項: なし

void Timer_SetTriggerMode(uint8 triggerMode)

機能: トリガモードを設定します。この関数は UDB 実装で、Trigger Mode パラメータが Software Controlled に設定された場合にのみ使用可能です。

パラメータ: uint8: 列挙型キャプチャ モード。コントロールレジスタセクションも参照してください:

Timer__B_TIMER__TM_NONE
Timer__B_TIMER__TM_RISINGEDGE
Timer__B_TIMER__TM_FALLINGEDGE
Timer__B_TIMER__TM_EITHEREDGE
Timer__B_TIMER__TM_SOFTWARE

返り値: なし

注意事項: なし

void Timer_EnableTrigger(void)

機能: トリガを有効にします。この関数は**Trigger Mode**が**Software Controlled**に設定されている場合のみ利用可能です。

パラメータ: なし

返り値: なし

注意事項: なし

void Timer_DisableTrigger(void)

機能: トリガを無効にします。この関数は**Trigger Mode**が**Software Controlled**に設定されている場合のみ利用可能です。

パラメータ: なし

返り値: なし

注意事項: なし

void Timer_SetInterruptCount(uint8 interruptCount)

機能: InterruptOnCaptureにより割り込みが発生する前に、カウントするキャプチャ数を設定します。この関数は InterruptOnCaptureCount が有効である場合にのみ利用可能です。

パラメータ: uint8 interruptCount: InterruptOnCapture による割り込みが発生する前に、カウントするキャプチャイベント数。0 ~ 3 の値が有効です。

返り値: なし

注意事項: なし

void Timer_ClearFIFO(void)

機能: キャプチャ FIFO をクリアします。この関数は UDB 実装でのみ使用可能です。このデータシートの 機能の詳細セクション内、UDB FIFO を参照してください。

パラメータ: なし

返り値: なし

注意事項: なし

void Timer_Sleep(void)

機能: これは、コンポーネントのスリープを準備するための、推奨されるルーチンです。Timer_Sleep() は、現在のコンポーネントの状態を保存します。その後、Timer_Stop() 関数を呼び出し、Timer_SaveConfig() を呼び出してハードウェア構成を保存します。

CyPmSleep() および CyPmHibernate()関数を呼び出す前に、Timer_Sleep()関数を呼び出してください。電源管理関数については、PSoC Creator *System Reference Guide* を参照してください。

パラメータ: なし

返り値: なし

注意事項: FF 実装では、全ての低消費電力モードで全レジスタの値が保持されます。UDB 実装では、コントロールレジスタとカウンタ値レジスタが保存、復元されます。また、Timer_Sleep()を呼び出す場合、Timer_Stop() を呼び出さずに Timer_Sleep を呼び出した場合に備えて、イネーブル状態が保存されます。

void Timer_Wakeup(void)

機能: これは、コンポーネントを Timer_Sleep() が呼び出されたときの状態に復元するのに推奨されるルーチンです。Timer_Wakeup() 関数は、設定を復元させるために Timer_RestoreConfig() 関数を呼び出します。Timer_Sleep 関数を呼び出す前にコンポーネントが有効であったならば、Timer_Wakeup() 関数もコンポーネントを再度有効にします。

パラメータ: なし

返り値: なし

注意事項: 最初に Timer_Sleep() または Timer_SaveConfig() 関数を呼び出すことなくTimer _Wakeup() 関数を呼び出すと、予期しない動作をする可能性があります。

void Timer_Init(void)

機能: Configureダイアログの設定に従って、コンポーネントを初期化または復元します。Timer_Start() ルーチンがこの関数を呼び出すので、Timer_Init() 関数を呼び出す必要はありません。これはコンポーネントの動作を開始する際に推奨される方法です。

パラメータ: なし

返り値: なし

注意事項: 全レジスタは、Configureダイアログの設定に従って、値が設定されます。

void Timer_Enable(void)

機能: ハードウェアを起動し、コンポーネントの処理を開始します。Timer_Start() ルーチンがこの関数を呼び出すので、Timer_Enable() 関数を呼び出す必要はありません。これはコンポーネントの動作を開始する際に推奨される方法です。この関数は、ソフトウェア制御のイネーブルモードに対し、タイマを有効にします。

パラメータ: なし

返り値: なし

注意事項: **Enable Mode** パラメーターが **Hardware Only**に設定されている場合、この関数はタイマの動作に何ら影響しません。

void Timer_SaveConfig(void)

機能: この機能により、コンポーネントの設定および非保持レジスタが保存されます。現在のコンポーネントパラメータ値も、Configureダイアログに定義されているように、または適切なAPIによって変更されているように、保存されます。この関数は、Timer_Sleep()関数に呼び出されます。

パラメータ: なし

返り値: なし

注意事項: なし

void Timer_RestoreConfig(void)

機能: この関数は、コンポーネントの設定と非保持レジスタを復元します。合わせてこの関数は、コンポーネントのパラメータ値をTimer_Sleep()関数を呼び出す前の状態に復旧します。

パラメータ: なし

返り値: なし

注意事項: Timer_Sleep()またはTimer_SaveConfig() 関数を呼び出す前に、この関数を呼び出した場合、予期しない動作をすることがあります。

ファームウェアソースコードのサンプル

PSoC Creator は、Find Example Project ダイアログに、回路図およびサンプルコードを含む多くのサンプルプロジェクトを提供しています。コンポーネント特有のサンプルを見るには、Component Catalog 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。一般的なサンプルについては、Start Page または **File** 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** 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 を参照してください。

機能の詳細

前述したとおり、タイマコンポーネントは複数の用途に設定できます。このセクションでは、これらの設定をより詳しく説明します。

動作概要

クロック入力の各立ち上がりエッジで、タイマコンポーネントは常にカウントダウンします。カウントが 0 に達したら、次のクロックエッジで、周期レジスタからカウンタレジスタへリロードします。

タイマは、ハードウェアかソフトウェアによって(構成設定によります)有効にされるまで無効のままです。Timer_Start() が呼び出されるまでコンポーネントは使用できません。それはこの関数がレジスタを定義された構成に設定するからです。

タイマ出力

カウンタレジスタをモニターしてリロードすることができます。カウンタレジスタの現在の値をモニターするために tc 出力が利用可能です。カウンタが 0 の間はハイレベルです。

タイマ入力

キャプチャ操作はハードウェアでもファームウェアでもどちらでも可能です。カウンタ レジスタの現在値はキャプチャ レジスタまたは FIFO にコピーされます。後でファームウェアがキャプチャされた値を読み取ります。

リセットおよびイネーブル機能を使って、タイマ コンポーネントを他のコンポーネントに同期できます。タイマ コンポーネントは有効にされているときにのみカウントし、リセット後は保持しません。カウントはトリガ入力イベントでも開始できます。リセットとイネーブルはハードウェアでもファームウェアでも可能です。トリガは全てハードウェアです。

注 FF タイマ実装での全ての入力 (キャプチャ、リセット、イネーブル) は FF タイマ内で二重同期です。シンクロナイザは BUS_CLK スピードで動作します。これにより、これらの信号が加えられてから、その効果が表れるまでの間に遅延が発生します。この遅延は、BUS_CLK とタイマを動かすクロックとの比率によります。FF 実装向けに示す全ての波形は、同期した後のシグナルを示します。

タイマ割り込み

割り込み出力を使って、イベントの発生を CPU または他のコンポーネントに伝えることができます。割り込みはイベントの組み合わせでアクティブになるように設定できます。割り込みハンドラは注意深く設計する必要があります。そうすれば、割り込みソースが何か、エッジとレベルのどちらを検出するか決めることができ、割り込みソースをクリアできます。

タイマレジスタ

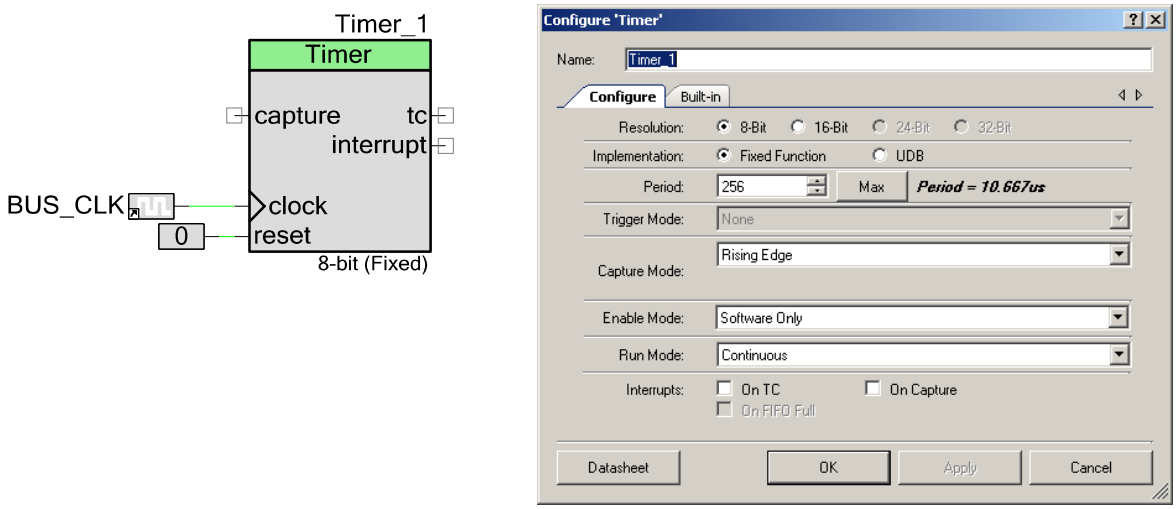
モード、ステータス、コントロールの 3 つのレジスタがあります。レジスタ セクションを参照してください。

設定

初期設定

タイマ コンポーネントを PSoC Creator の回路図にドラッグしたときの初期設定は、クロック入力の立ち上がりエッジでカウンタレジスタをデクリメントする 8 ビット、FF タイマです。図 1 は初期設定の回路図マクロと Configure ダイアログ設定を示しています。

図 1. タイマの初期設定



このタイマの細かな機能は、実装やシリコンで異なります。以後の図は、UDB 実装されたタイマと異なるシリコン上での FF 実装されたタイマの機能を示しています。

UDB 実装で設定した場合の初期設定での機能を 図 2 に示します。

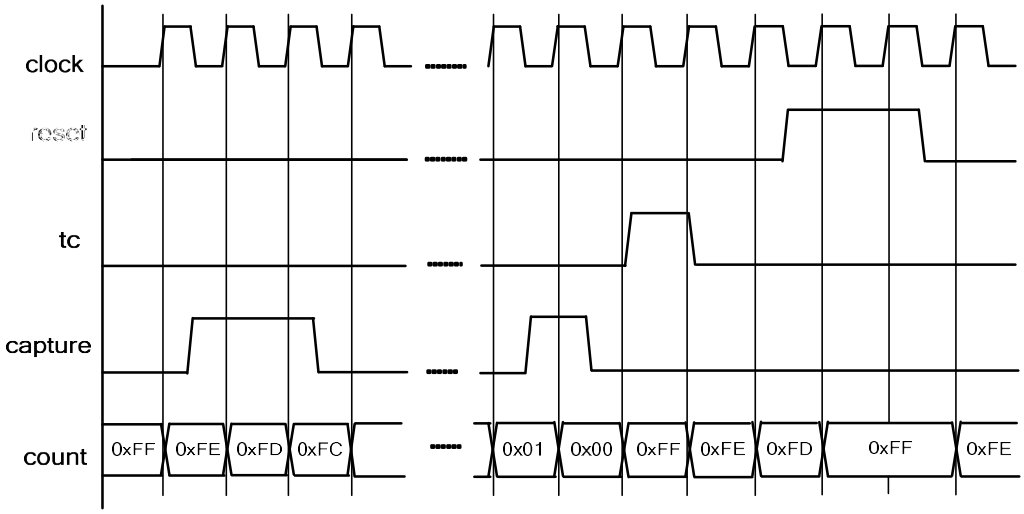
カウンタはタイマ設定中にプリロードされ、カウントが 0 に達するごとにリロードされます。初期設定では、**Period** は 256 に設定されています。これにより、0xFF がカウンタにロードされます。なぜならば、0xFF から 0 までカウントすると 周期 256 を算出するからです。

リセット信号は周期レジスタからカウンタを強制的にリロードします。リセット信号が解除されるまで、カウンタはこの状態を保持します。

ターミナルカウントは、タイマが 0 までカウントしたことを示します。カウントが 0 に達したクロックサイクルに続くクロックサイクル上でアクティブです。ターミナルカウント信号は、リセットイベントに基づいて発生しません。

初期設定では、キャプチャ入力の立ち上がりエッジ毎にキャプチャするように、キャプチャ機能が設定されています。キャプチャパルスの幅に関わらず、一つの値がキャプチャされます。この例では、0xFE と 0x01 の値をキャプチャして CPU が読み込むことができます。

図 2. 初期設定の UDB タイマ実装による波形例



PSoC 3 の専用機能ブロック実装で設定した初期設定の機能を図 3 に示します。

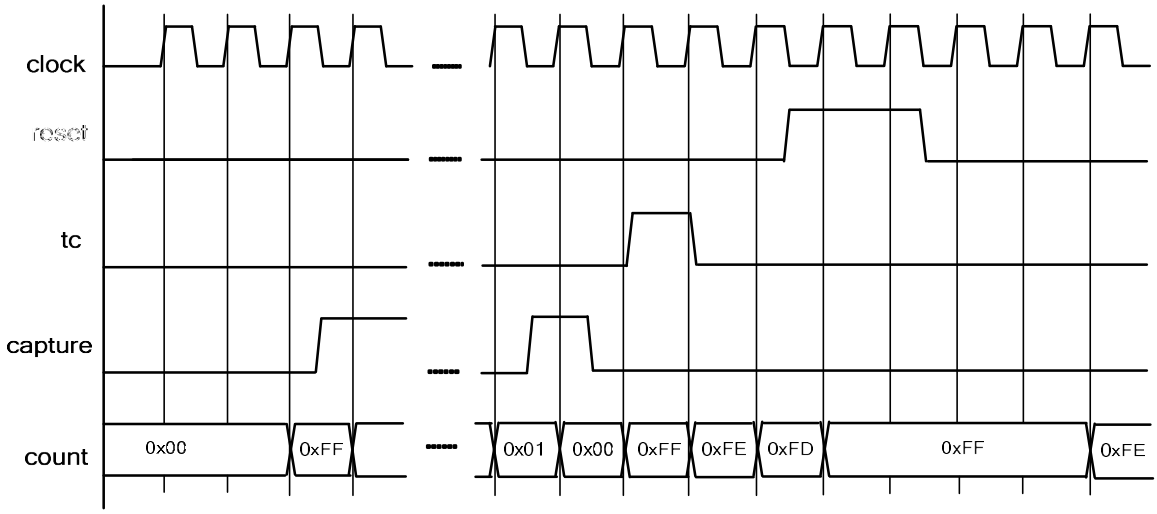
専用機能ブロック実装では、カウンタ値は設定時間中にプレロードされません。そのかわり、カウントは 0 から開始します。PSoC 3 では、これにより FF 実装と UDB 実装との間に、初期に 3 サイクルの遅延を生じます。タイマがカウントを開始する前は、時間のずれは 2 サイクルで、周期レジスタからカウンタをロードする際には 1 サイクルです。タイマの稼働後は、UDB 実装と周期は同じになります。

リセット信号は周期レジスタからカウンタを強制的にリロードし、リセット信号が解除されるまで、カウンタはこの状態を保持します。リセットが解除されると、カウンタがカウントダウンを開始する前に 2 サイクルの遅延があります。

ターミナルカウントは、タイマが 0 までカウントしたことを示します。カウントが 0 に達したクロックサイクルに続くクロックサイクル上でアクティブです。ターミナルカウント信号は、リセットイベントまたは初期のカウントが 0 であることに基づいては発生しません。

初期設定では、キャプチャ入力の立ち上がりエッジ毎にキャプチャするように、キャプチャ機能が設定されています。キャプチャパルスの幅に関わらず、一つの値がキャプチャされます。この例では、0xFF と 0x01 の値をキャプチャして、CPU が読み込むことができます。この機能は UDB 実装と同じです。

図 3. 初期設定の PSoC 3 FF タイマ実装による波形例



PSoC 5 の専用機能ブロック実装で設定した場合の、初期設定の機能を 図 4 に示します。

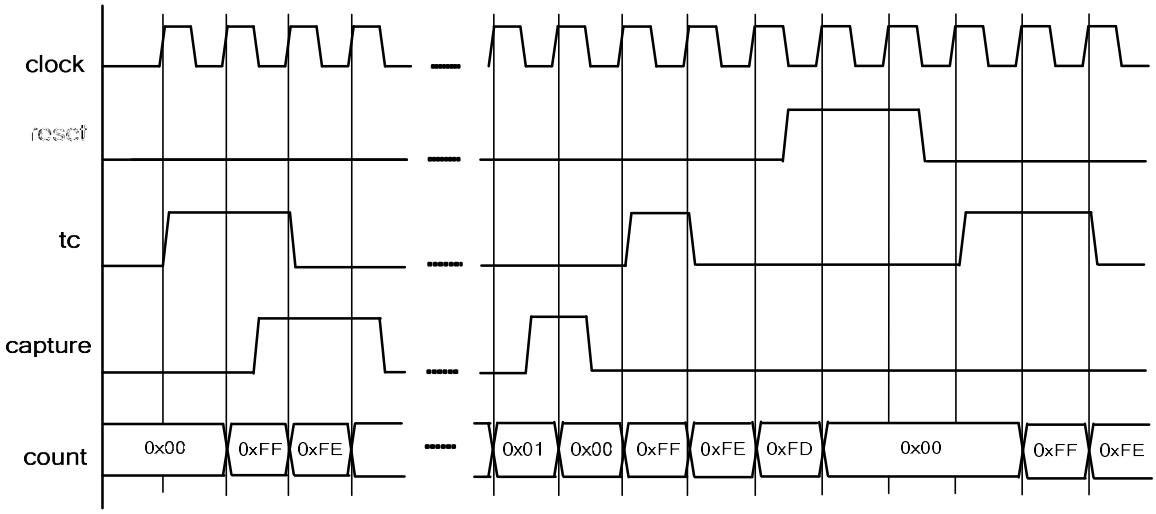
専用機能ブロック実装では、カウントは設定時間中にプレロードされません。そのかわり、カウントは 0 から開始します。PSoC 5 では、これにより FF 実装と UDB 実装との間に、初期に 2 サイクルの遅延を生じます。タイマがカウントを開始する前は、時間のずれは 1 サイクルで、周期レジスタからカウンタをロードする際には 1 サイクルです。タイマの稼働後は、UDB 実装と周期は同じになります。

リセット信号はカウンタを強制的にクリアし、リセット信号が解除されるまで、カウンタはこの状態を保持します。リセット後の機能は、UDB 実装より最初の周期が 2 サイクル長い初期状態をもつものになります。

ターミナルカウントは、タイマが 0 までカウントしたことを示します。カウンタの初期値とリセット時の値を組み合わせると、初期化時とリセット後に 2 サイクルの TC パルスを生じます。リセットがアクティブな間 TC はローレベルに維持されますが、リセットが解除された後は 2 サイクルの間ハイレベルになります。

初期設定では、キャプチャ入力の立ち上がりエッジ毎にキャプチャするように、キャプチャ機能が設定されています。キャプチャパルスの幅に関わらず、一つの値がキャプチャされます。この例では、0xFF と 0x01 の値をキャプチャして、CPU が読み込むことができます。この機能は、UDB 実装と同じです。

図 4. 初期設定の PSoC 5 FF タイマ実装による波形例

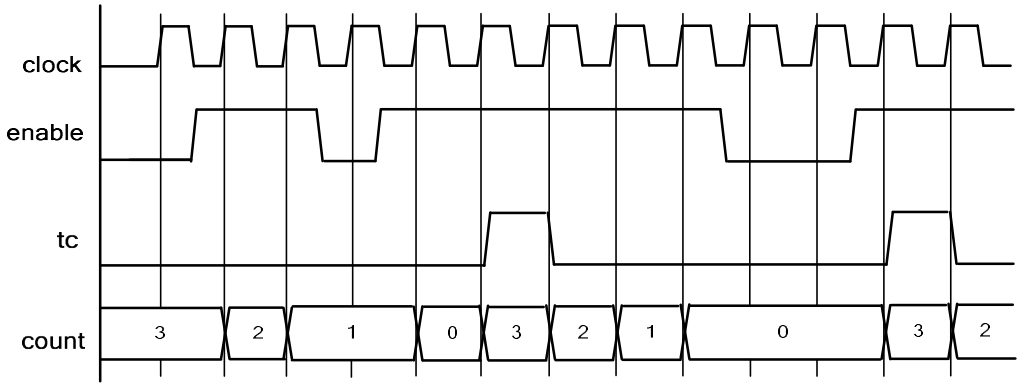


ソフトウェアおよびハードウェアのイネーブル設定

ハードウェアイネーブルの機能は特定の実装に基づいて異なります。UDB 実装のソフトウェアおよびハードウェアイネーブルに設定した場合のタイマの機能を 図 5 に示します。

タイマが有効の場合は、カウンタはクロックサイクル毎にデクリメントします。サイクル中にカウンタが周期レジスタからリロードされた場合は、単一サイクルのターミナルカウントパルスが発生します。TC 信号は常に単一クロックサイクルのパルスです。リロードサイクル中に発生することに注意してください。カウントが 0 になったためにカウンタが無効になりリロードが遅れた場合は、カウンタが再度有効になりリロードされるまで、TC パルスも遅れます。リセット信号のためにカウンタが強制的にリロードされると、TC パルスは発生しません。

図 5. SW および HW イネーブルの UDB タイマ実装による波形例

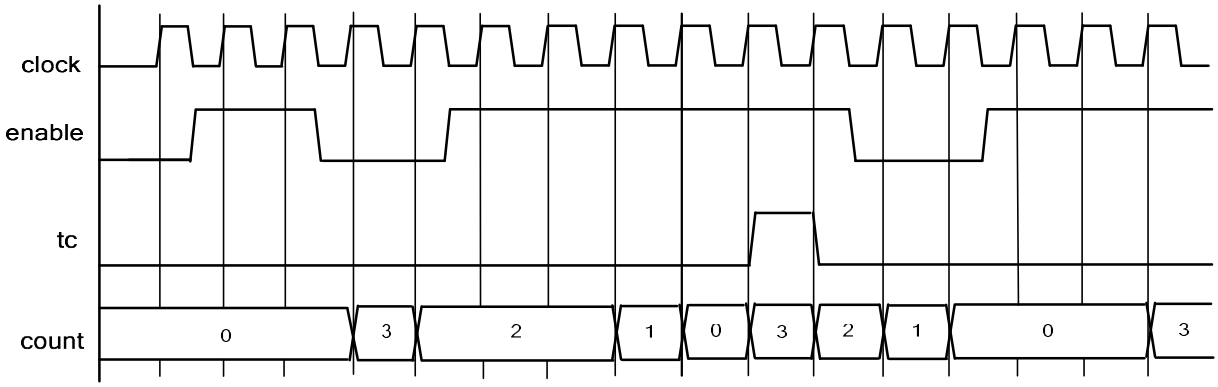


PSoC 3 FF 実装でソフトウェアおよびハードウェアイネーブルに設定した場合のタイマの機能を 図 6 に示します。

カウンタのハードウェアイネーブルからイネーブルが有効になるまで 2 クロックサイクルの遅延があります。その結果、2 クロックサイクル前のイネーブル信号がハイレベルだと、カウンタはデクリメントします。この遅延は、カウンタをイネーブルにするときにも、ディスエーブルにするときにも適用されます。サイクル中にカウンタが周期レジスタからリロードされた場合は、単一サイクルのターミナルカウントパルスが発生します。TC 信号は常に単一クロックサイクルのパルスです。

注 カウントが 0 になる前 2 クロックサイクルの間に、タイマのイネーブル信号がローレベルになると、TC 出力パルスはタイマのその周期には発生しません。タイマが再び有効にされると、TC 信号を発生せずリロードされます。波形の例に示します。

図 6. SW および HW イネーブルの PSoC 3 FF タイマ実装による波形例

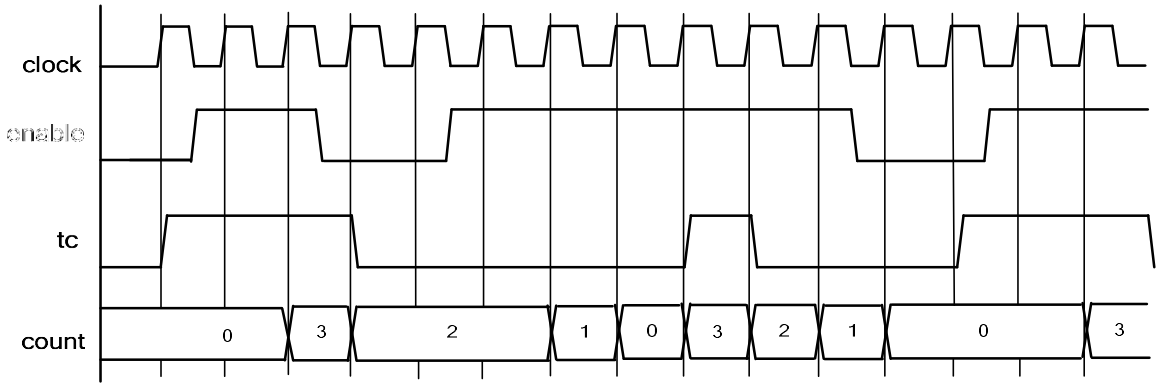


PSoC 5 FF 実装でソフトウェアおよびハードウェアイネーブルに設定した場合のタイマの機能を 図 7 に示します。

カウンタのハードウェアイネーブルからイネーブルが有効になるまで 1 クロックサイクルの遅延があります。その結果、1 クロックサイクル前のイネーブル信号がハイレベルだと、カウンタはデクリメントします。この遅延は、カウンタをイネーブルにするときにも、ディスエーブルにするときにも適用されます。カウントが 0 に等しいときはいつでも、1 クロックサイクルの遅延でターミナルカウント信号が発生します。これは最初の設定時に発生します。カウントが 0 の間にイネーブル信号がタイマを停止すると、TC 信号はハイレベルのままです。

注 イネーブル信号が単一サイクル時間イナクティブのパルスを発する場合、ハードウェアイネーブル信号は予期したようには機能しません。単一サイクルのディスエーブルパルスは、タイマが再び無効になってから次に有効にされるまで、タイマをそのカウントでロックします。このため、ハードウェアディセーブルは常に 2 サイクル以上である必要があります。単一サイクルのイネーブルは予期したように機能します。

図 7. SW および HW イネーブルの PSoC 5 FF タイマ実装による波形例



ワンショット設定

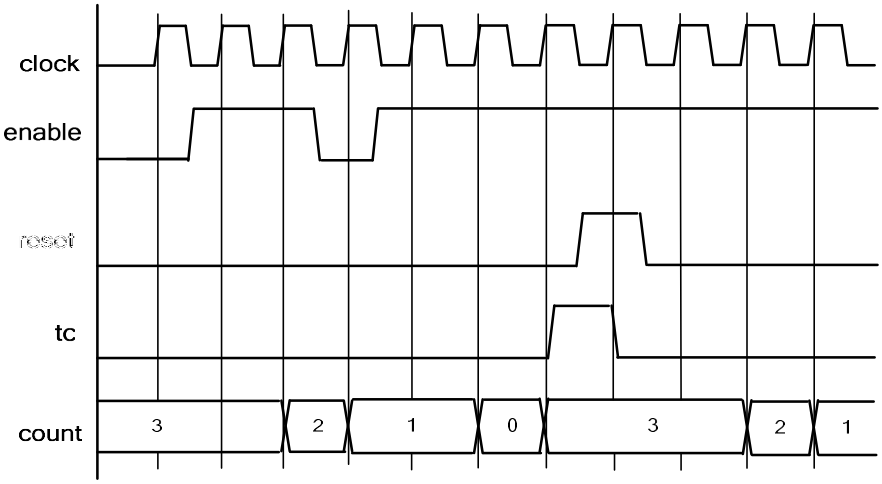
ワンショット動作モードの機能は特定の実装に基づいて異なります。UDB 実装の One Shot 動作に設定した場合のタイマの機能を 図 8 に示します。

カウンタのハードウェアイネーブルからイネーブルが有効になるまで 1 クロックサイクルの遅延があります。その結果、1 クロックサイクル前のイネーブル信号がハイレベルだと、カウンタはデクリメントします。この遅延は、カウンタをイネーブルにするときにも、ディスエーブルにするときにも適用されます。これは、遅延なしにカウントする Continuous 動作モードとは異なる動作です。

TC 信号は常に単一クロックサイクルのパルスです。リロードサイクル中に発生することに注意してください。カウンタがゼロカウントになったので無効となり、リロードが遅れた場合は、カウンタが再度有効になりリロードされるまで、TC パルスも遅れます。リセット信号のためにカウンタが強制的にリロードすると、TC パルスは発生しません。

ワンショット周期の完了後、ハードウェアをリセットすることでタイマをさらにもう 1 周期実行するように設定できます。ハードウェア リセットは周期レジスタからカウンタをリロードします。リセット解除されると 1 サイクル後に、タイマはハードウェアもアクティブになった後、再度カウントを開始できるようになります。

図 8. One Shot 動作の UDB タイマー実装による波形例



PSoC 3 で FF 実装の One Shot 動作に設定した場合のタイマの機能を 図 9 に示します。

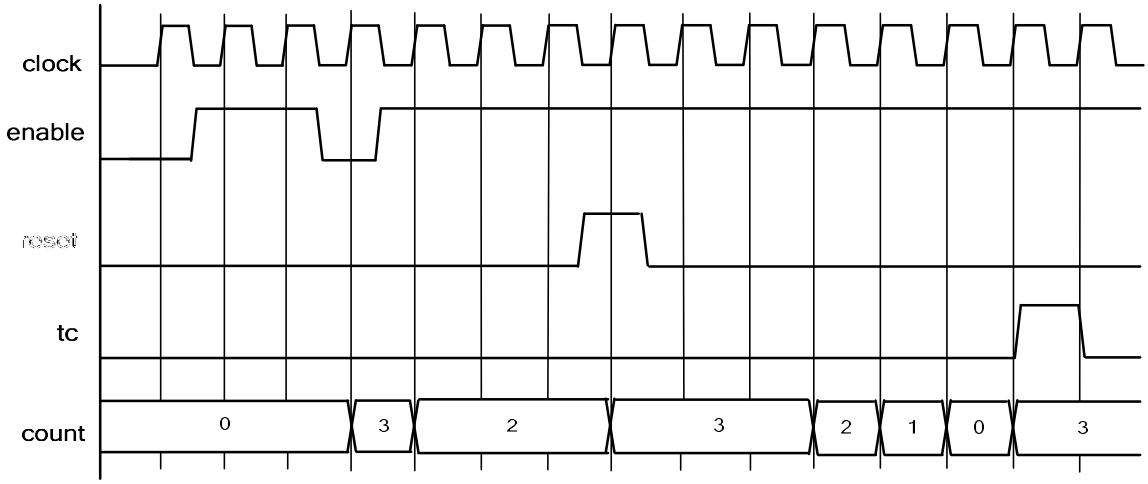
カウンタのハードウェアイネーブルからイネーブルが有効になるまで 2 クロックサイクルの遅延があります。その結果、2 クロックサイクル前のイネーブル信号がハイレベルだと、カウンタはデクリメントします。この遅延は、カウンタをイネーブルにするときにも、ディスエーブルにするときにも適用されます。サイクル中にカウンタが周期レジスタからリロードされた場合は、単一サイクルのターミナルカウントパルスが発生します。TC 信号は常に単一クロックサイクルのパルスです。これは Continuous 動作モードの動作と全く同じです。

この実装でのみですが、One Shot モードの特別な機能として、タイマがカウントを開始し、イネーブル信号が初めてローレベルになると、その値でカウンタを停止します。再度カウントを開始するためには、タイマをリセットする必要があります。

One Shot 周期の完了後、またはイネーブル信号がディスエーブルになったために停止した後、ハードウェアをリセットしてタイマをさらに 1 周期実行するように設定できます。ハードウェア リセットは周期レジスタからカウンタをリロードします。リセットを解除してから、タイマが再度カウントダウンできるようになるまでの間に、2 サイクルの遅延があります。

注 この実装では、タイマは、Timer_Stop() API の後に Timer_Start() API を使用してのみ再開できます。これにより、カウンタは引き続きカウントできますが、カウントをリロードしません。そのため、カウンタが 1 周期終了しリロードされた場合にのみこの方法を使うべきです。

図 9. One Shot 動作の FF PSoC 3 タイマー実装による波形例



PSoC 5 で FF 実装の One Shot ット動作に設定した場合のタイマの機能を 図 10 に示します。

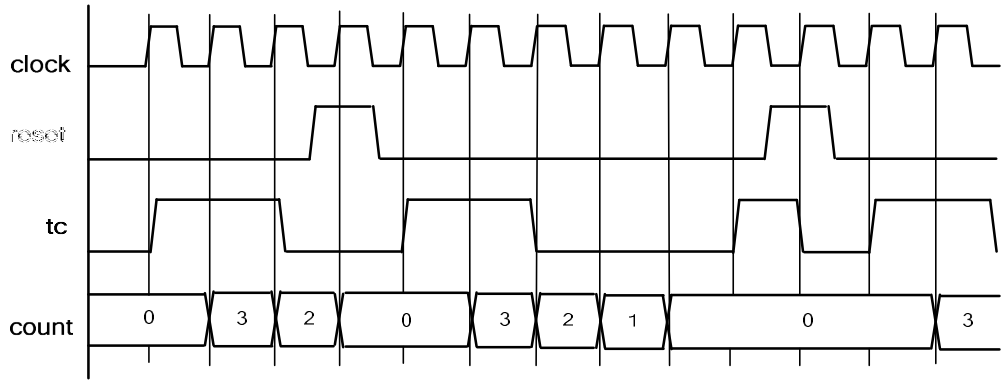
カウントが 0 に等しいときはいつでも、1 クロックサイクルの遅延でターミナルカウント信号が発生します。これは最初の設定時に発生します。TC 信号は、1 つのワンショット周期が完了した後でハイレベルにとどまります。カウントが 0 にとどまるからです。カウントが 0 の場合の、TC 信号発生の例外の 1 つは、リセット信号がアクティブな場合に TC は常に 0 に保持されます。

One Shot 周期の完了後、ハードウェアをリセットしてタイマをさらに 1 周期実行するように設定できます。ハードウェアリセットはカウント 0 でリロードし、タイマを再度実行するように設定します。リセットを解除してから、タイマが再度カウントダウンできるようになるまでの間に、1 サイクルの時間の遅延があります。

注 ハードウェアイネーブルのワンショットモードは PSoC 5 FF 設定に対応していません。

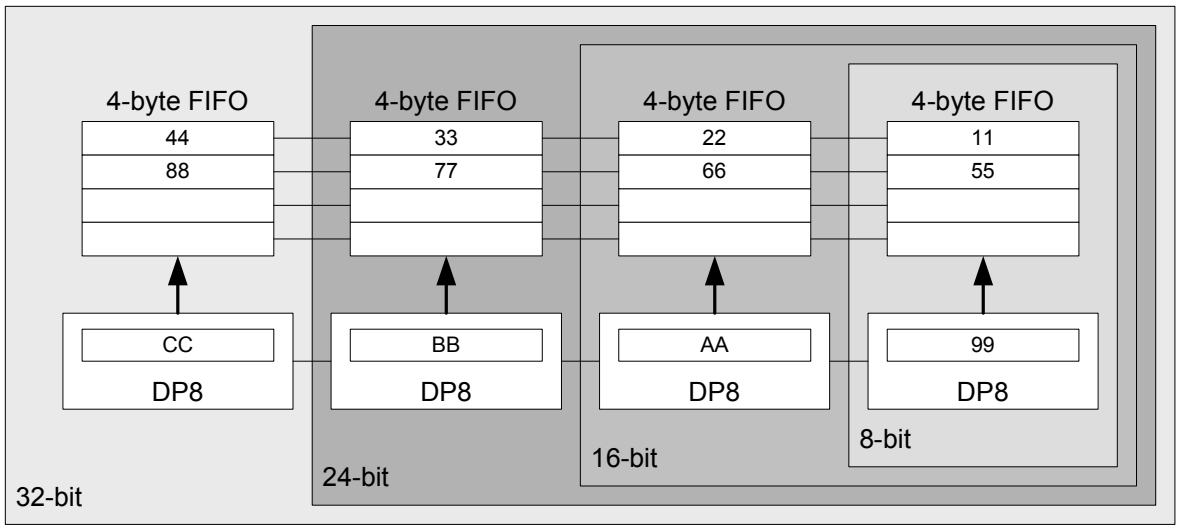
注 リセット信号がアクティブな場合に TC 信号がローレベルに保持されるので、ワンショットを実行し完了するまでの間に、2 つの TC パルスが各々発生します。第一は、カウンタが 0 をカウントする時です。第二は、リセットが解除された後、カウンタがカウントを開始する前です。波形の例に示します。

図 10. One Shot 動作の FF PSoC 5 タイマ実装の波形例



UDB FIFO

UDB データパス FIFO はカウントをキャプチャするために使用されます。各 FIFO は 4 バイトです。マルチバイト設定では、カウンタの各バイトが関連する UDB の FIFO に同時にキャプチャされます。このため、データの喪失を防ぐために CPU がキャプチャレジスタを読み込まなければならなくなるまでに 4 回分のキャプチャまで可能です。



Capture Value #1 = 0x44332211
Capture Value #2 = 0x88776655
Accumulator = 0xCCBBAA99

レジスタ

ステータスレジスタ

ステータスレジスタは、タイマに定義される状態ビットを含む読み取り専用レジスタです。Timer_ReadStatusRegister() 関数を使って、ステータスレジスタ値を読み取ります。これらのビットフィールドは FF と UDB 実装間で異なる可能性があるので、ステータスレジスタ上のすべての演算は、ビットフィールドに対して次の定義を使用する必要があります。

ステータスレジスタの一部のビットはスティッキーです。つまり 1 にセットした後、レジスタが読まれてクリアされるまでその状態を保持します。ステータスデータはタイマの入力クロックのエッジでレジスタに取り込まれ、全スティッキービットにタイマのタイミング分解能を与えます。すべての非スティッキービットは参照可能で、入力から直接ステータスレジスタに読み込まれます。

Timer_Status (UDB 実装)

ビット	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	FIFO Not Empty	FIFO Full	Capture	TC
スティッキー	該当せず	該当せず	該当せず	該当せず	偽	偽	真	真

Timer_Status (FF 実装)

ビット	7	6	5	4	3	2	1	0
名称	TC	Capture	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD
スティッキー	真	真	該当せず	該当せず	該当せず	該当せず	該当せず	該当せず

ビット名	ヘッダファイルの定義	説明
TC	Timer_STATUS_TC	このビットはカウントが0に等しいときに 1 になります。
Capture	Timer_STATUS_CAPTURE	このビットは有効なキャプチャイベントがトリガされると 1 になります。これはソフトウェア キャプチャを含みません。
FIFO Full	Timer_STATUS_FIFOFULL	このビットは UDB FIFO が 4 エントリとして定義されたフル状態に達したときに 1 になります。
FIFO Not Empty	Timer_STATUS_FIFONEMP	このビットは UDB FIFO に少なくとも 1 つエントリがあるときに 1 になります。

モードレジスタ

モードレジスタは読み取り/書き込みレジスタで、カウンタ用に定義された割り込みマスクを含みます。Timer_SetInterruptMode() 関数を使ってモードビットを設定します。これらのビットフィールドは FF と UDB 実装間で異なる可能性があるので、モードレジスタ上のすべての演算は、ビットフィールドに対して次の定義を使用する必要があります。

タイマコンポーネントの割り込み出力は全割り込みソースの論理和関数です。各ソースはモードレジスタ内の対応ビットでイネーブルまたはマスクできます。

Timer_Mode (UDB 実装)

ビット	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	RSVD	FIFO Full	Catpure	TC

Timer_Mode (FF 実装)

ビット	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	TC	Capture	RSVD	RSVD

ビット名	ヘッダファイルの定義	割り込み出力オン
TC	Timer_STATUS_TC_INT_MASK	カウンタレジスタが 0 に等しい
Capture	Timer_STATUS_CAPTURE_INT_MASK	Capture
FIFO Full	Timer_STATUS_FIFOFULL_INT_MASK	UDB FIFO がフル

コントロールレジスタ

コントロールレジスタを使うと、カウンタの全体的動作を制御できます。このレジスタは Counter_WriteControlRegister() 関数で書き込まれ、Counter_ReadControlRegister() 関数で読み取られます。これらのビットフィールドは FF と UDB 実装間で異なる可能性があるので、コントロールレジスタ上のすべての演算は、ビットフィールドに対して次の定義を使用する必要があります。

注 コントロールレジスタに書き込むときは予約ビットはどれも変更できません。すべての動作は、予約ビットをマスクした読み取り-変更-書き込みでなければなりません。

Timer_Control (UDB Implementation)

ビット	7	6	5	4	3	2	1	0
名称	Enable	Capture Mode [1:0]		Trigger Enable	Trigger Mode [1:0]		Interrupt Count [1:0]	

Timer_Control1 (FF 実装)

ビット	7	6	5	4	3	2	1	0
名称	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	Enable

ビット名	ヘッダファイルの定義	説明/列挙型
Interrupt Count	Timer_CTRL_INTCNT_MASK	割り込みカウントビットは、割り込みが発生する前にカウントするキャプチャイベントの数を定義します。
Trigger Mode	Timer_CTRL_TRIG_MODE_MASK	トリガモード制御ビットは、所望のトリガ入力機能を定義します。このビットは初期化時に **Trigger Mode**パラメーターで定義したトリガモードで設定します。 ■ Timer__B_TIMER__TM_NONE ■ Timer__B_TIMER__TM_RISINGEDGE ■ Timer__B_TIMER__TM_FALLINGEDGE ■ Timer__B_TIMER__TM_EITHEREDGE ■ Timer__B_TIMER__TM_SOFTWARE
Trigger Enable	Timer_CTRL_TRIG_EN	トリガイネーブルビットは、トリガ·イベントの準備をするときのソフトウェア制御を可能にします。
Capture Mode	Timer_CTRL_CAP_MODE_MASK	キャプチャ モード制御ビットは、所望のキャプチャ入力動作を定義するのに使用する 2ビット フィールドです。このビット フィールドは、**Capture Mode** パラメータで定義されたキャプチャモードで初期化時に設定されます。 ■ Timer__B_TIMER__CM_NONE ■ Timer__B_TIMER__CM_RISINGEDGE ■ Timer__B_TIMER__CM_FALLINGEDGE ■ Timer__B_TIMER__CM_EITHEREDGE ■ Timer__B_TIMER__CM_SOFTWARE
Enable	Timer_CTRL_ENABLE	ソフトウェア制御下でのカウントを有効にします。このビットは**Enable Mode** パラメーターが **Software Only** または **Software and Hardware**に設定されているときにのみ有効です。

カウンタ (8、16、24、32 ビット)

カウンタレジスタには現在のカウントが入っています。全てのクロック入力の立ち上がりエッジに対応して、このレジスタはデクリメントします。このレジスタはいつでも Timer_ReadCounter() 関数を呼び出すことで読み取ることができます。

キャプチャ (8、16、24、32 ビット)

キャプチャレジスタにはキャプチャしたカウントが入っています。キャプチャイベントはどれも、カウンタレジスタをこのレジスタにコピーします。UDB 実装では、このレジスタは実際に FIFO です。詳細については UDB FIFO セクションを参照してください。

周期 (8、16、24、32 ビット)

周期レジスタは、Timer_WritePeriod()関数呼び出しで設定し、初期化時に **Period** パラメータによって定義される周期を格納しています。周期レジスタはリロードイベントでカウンタレジスタにコピーされます。

コンポーネントのデバッグウィンドウ

タイマコンポーネントは、PSoC Creator コンポーネントのデバッグウィンドウをサポートします。次のレジスタは、デバッグウィンドウに表示されます。*で表示したいくつかのレジスタは、UDB 実装で利用可能であり、**で表示したいくつかのレジスタは、FF 実装でのみ利用可能です。その他のレジスタは、いずれの実装でも利用可能です。

レジスタ: Timer_CONTROL

名称: コントロールレジスタ

機能: ビットフィールドの定義については、このデータシートで前述した Timer_Control レジスタの説明を参照してください。

レジスタ: Timer_CONTROL2 **

名称: FFコントロールレジスタ #2

機能: FFタイマブロックには、第2の設定レジスタがあります。ビットフィールドの定義については、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。

レジスタ: Timer_STATUS_MASK *

名称: ステータスレジスタ割り込みマスク設定

機能: 任意の状態ビットを、コンポーネントの割り込み出力ピンに対する割り込みソースとして有効にできます。ビットフィールドの定義については、このデータシートで前述した Timer_Status レジスタの説明を参照してください。

レジスタ: Timer_STATUS_AUX_CTRL *

名称: ステータスレジスタの補助コントロールレジスタ

機能: ビットフィールド「INT_EN」を介して、内部ステータスレジスタの割り込み出力を有効にできます。ビットフィールドの定義については、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。

レジスタ: Timer_PERIOD

名称: タイマ周期レジスタ

機能: タイマの各周期の最初に周期カウンタへリロードされる周期を定義します。

レジスタ: Timer_COUNTER

名称: タイマカウンタレジスタ

機能: 現在のタイマ周期の (**Period** から 0 までのクロック数で) 現在のカウントを示します。

レジスタ: Timer_GLOBAL_ENABLE **

名称: FFタイマのグローバルイネーブルレジスタ

機能: FFタイマの動作を有効にします。ビットフィールドの定義については、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。

DC/ AC 電気的特性(FF 実装)

以下の値は、期待される性能を示しており、初期特性データを基にしています。

タイマ DC 仕様

記号	項目	条件	Min	Typ	Max	単位
	ブロック消費電流	16ビットタイマ、各入力クロック周波数にて	–	–	–	µA
	3MHz		–	15	–	µA
	12MHz		–	60	–	µA
	48MHz		–	260	–	µA
	67MHz		–	350	–	µA

タイマ AC 仕様

記号	項目	条件	Min	Typ	Max	単位
	動作周波数		DC	–	67	MHz
	キャプチャパルス幅(内部)		15	–	–	ns
	キャプチャパルス幅(外部)		30	–	–	ns
	タイマ分解能		15	–	–	ns
	イネーブルパルス幅		15	–	–	ns
	イネーブルパルス幅(外部)		30	–	–	ns
	リセットパルス幅		15	–	–	ns
	リセットパルス幅(外部)		30	–	–	ns

PSoC 5 DC/ AC 電気的特性(FF 実装)

以下の値は、期待される性能を示しており、初期特性データを基にしています。

タイマ DC 仕様

記号	項目	条件	Min	Typ	Max	単位
	ブロック消費電流	16ビットタイマ、各入力クロック周波数にて	–	–	–	µA
	3MHz		–	65	–	µA
	12MHz		–	170	–	µA
	48MHz		–	650	–	µA
	67MHz		–	900	–	µA

タイマ AC 仕様

記号	項目	条件	Min	Typ	Max	単位
	動作周波数		DC	–	67.01	MHz
	キャプチャパルス幅(内部)		13	–	–	ns
	キャプチャパルス幅(外部)		30	–	–	ns
	タイマ分解能		13	–	–	ns
	イネーブルパルス幅		13	–	–	ns
	イネーブルパルス幅(外部)		30	–	–	ns

	リセットパルス幅		13	–	–	ns
	リセットパルス幅(外部)		30	–	–	ns

DC/ AC 電気的特性(UDB 実装)

以下の値は、期待される性能を示しており、初期特性データを基にしています。

"公称配線の最大値"タイミング特性

記号	項目	構成	Min	Typ	Max	単位
f_{CLOCK}	コンポーネント クロック周波数	8ビット UDB タイマ	–	–	40	MHz
		16ビット UDBタイマ	–	–	38	MHz
		24ビット UDBタイマ	–	–	33	MHz
		32ビット UDBタイマ	–	–	27	MHz
t_{clockH}	入力クロックハイレベル時間 [4]	該当せず	–	0.5	–	t_{CY_clock}
t_{clockL}	入力クロックローレベル時間[4]	該当せず	–	0.5	–	t_{CY_clock}
入力						
t_{PD_ps}	入力配線遅延、pin to sync [5]	1	–	–	STA [6]	ns
t_{PD_ps}	入力配線遅延、pin to sync	2	–	–	8.5	ns
t_{PD_si}	Sync出力から入力への遅延(配線) [5]	1,2,3,4	–	–	STA [6]	ns
t_{I_clk}	clockX と clock のアライメント	1,2,3,4	0	–	1	t_{CY_clock}
t_{PD_IE}	コンポーネントクロックへの入力配線遅延(エッジ検出入力)	1,2	t_{PD_ps} + t_{SYNC} + t_{PD_si}	–	t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}	ns
t_{PD_IE}	コンポーネントクロックへの入力配線遅延(エッジ検出入力)	3,4	t_{SYNC} + t_{PD_si}	–	t_{SYNC} + t_{PD_si} + t_{I_clk}	ns

[4] t_{CY_clock} = 1/f_{CLOCK}. これは 1 クロック周期のサイクル時間です。

[5] t_{PD_ps} および t_{PD_si} は配線経路の遅延。配線は動的なためこれらの値は変化することがあり、最大コンポーネントクロックと同期クロック周波数に直接の影響を及ぼします。これらの値は、Static Timing Analysis Results(静的タイミング解析結果)に記載されています。

[6]構成 2 における t_{PD_ps} は、デバイスのピン毎に定義された固定値です。ここに挙げた数字は、デバイスで利用可能なすべてのピンの公称値です。

記号	項目	構成	Min	Typ	Max	単位
t_{IH}	入力ハイレベル時間	1,2,3,4	t_{CY_clock}	–	–	ns
t_{IL}	入力ローレベル時間	1,2,3,4	t_{CY_clock}	–	–	ns

"全配線の最大値"タイミング特性

記号	項目	構成	Min	Typ	Max [7]	単位
f_{CLOCK}	コンポーネント クロック周波数	8ビット UDB タイマ	–	–	20	MHz
		16ビット UDBタイマ	–	–	15	MHz
		24ビット UDBタイマ	–	–	20	MHz
		32ビット UDBタイマ	–	–	15	MHz
t_{clockH}	入力クロックハイレベル時間 [8]	該当せず	–	0.5	–	$1/f_{clock}$
t_{clockL}	入力クロックローレベル時間	該当せず	–	0.5	–	$1/f_{clock}$
Inputs (入力)						
t_{PD_ps}	入力配線遅延、pin to sync [9]	1	–	–	STA	ns
t_{PD_ps}	入力配線遅延、pin to sync [10]	2	–	–	8.5	ns
t_{PD_si}	Sync出力から入力への遅延(配線) [9]	1,2,3,4	–	–	STA [9]	ns
t_{I_clk}	clockX と clock のアライメント	1,2,3,4	0	–	1	t_{CY_clock}
t_{PD_IE}	コンポーネントクロックへの入力配線遅延(エッジ検出入力)	1,2	$t_{PD_ps} + t_{SYNC} + t_{PD_si}$	–	$t_{PD_ps} + t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	コンポーネントクロックへの入力配線遅延(エッジ検出入力)	3,4	$t_{SYNC} + t_{PD_si}$	–	$t_{SYNC} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	入力ハイレベル時間	1,2,3,4	t_{CY_clock}	–	–	ns
t_{IL}	入力ローレベル時間	1,2,3,4	t_{CY_clock}	–	–	ns

[7] 全配線の最大値タイミング特性は、公称配線の最大値タイミング特性 を 2 倍に緩和して計算します。お使いのコンポーネントインスタンスがこの速度と同じ、または遅いスピードで動作している場合は、タイミングはこのコンポーネントで問題にはなりません。

[8] $t_{CY_clock} = 1/f_{CLOCK}$. これは 1 クロック周期のサイクル時間です。

[9] t_{PD_ps} および t_{PD_si} は配線経路の遅延。配線は動的なためこれらの値は変化することがあり、最大コンポーネントクロックと同期クロック周波数に直接の影響を及ぼします。これらの値は、Static Timing Analysis Results(静的タイミング解析結果)に記載されています。

[10]構成 2 における t_{PD_ps} は、デバイスのピン毎に定義された固定値です。ここに挙げた数字は、デバイスで利用可能なすべてのピンの公称値です。

特性データの STA 項目の見方

公称配線最大値は、静的タイミング解析 (STA) による複数のテストパスから収集されます。STA の結果から次の手法で設計の最大値を計算できます。

f_{clock} 最大コンポーネントクロック周波数が、名称の付いた外部クロックとしてタイミング結果のクロックサマリに表示されます。下図は、*_timing.html* から抜粋したクロック制限の例を示しています。

-Clock Summary

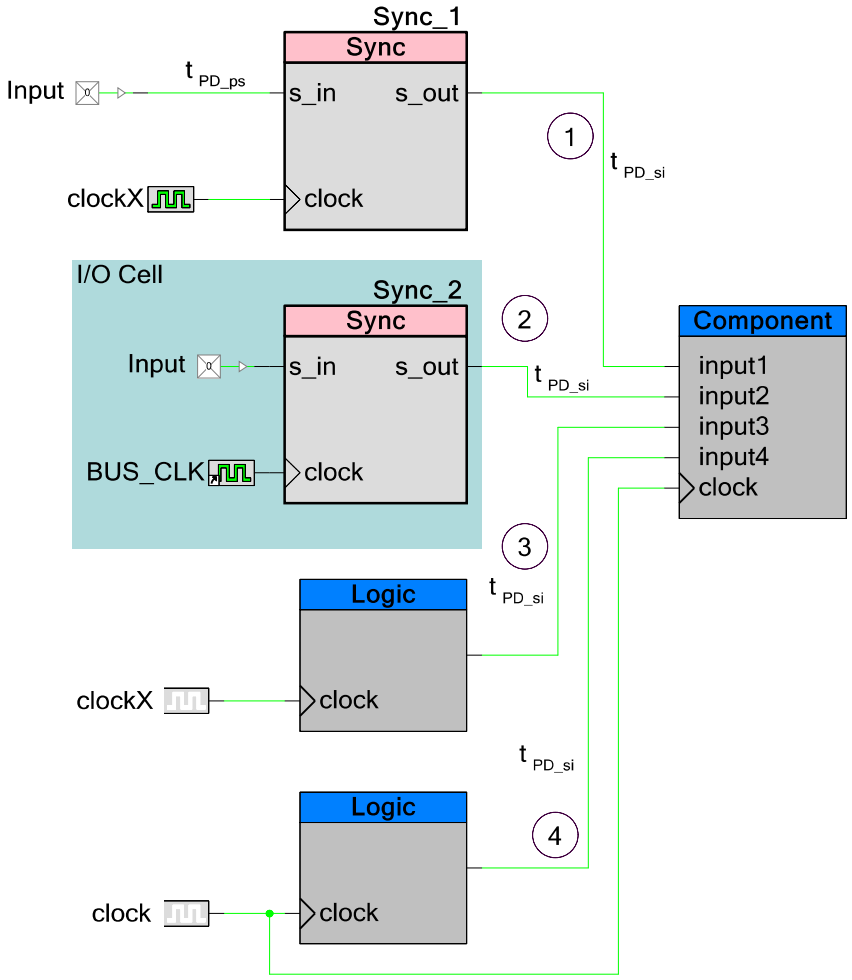
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

入力配線遅延とパルス幅

入力の機能解析を行う場合、すべての入力は、大きく図 11 に示す 4 つの構成の内 1 つに該当します。

すべての入力は同期されていなければなりません。同期のメカニズムは、コンポーネントへの入力ソースによって異なります。システムの動作を完全に解釈するには、各入力を構成した設定とシステムのクロック構成を理解する必要があります。このセクションでは、Static Timing Analysis (静的タイミング解析、STA) の結果を使用して、システムの特性解析を行う方法について説明します。

図 11. コンポーネントタイミング仕様のための入力構成



構成	コンポーネントクロック	シンクロナイザクロック(周波数)	図
1	master_clock	master_clock	図 16
1	clock	master_clock	図 14
1	clock	clockX = clock [11]	図 12
1	clock	clockX > clock	図 13
1	clock	clockX < clock	図 15
2	master_clock	master_clock	図 16
2	clock	master_clock	図 14

[11] クロック周波数は同じですが、立ち上がりエッジのアライメントは保証されていません。

構成	コンポーネントクロック	シンクロナイザクロック(周波数)	図
3	master_clock	master_clock	図 21
3	clock	master_clock	図 19
3	clock	clockX = clock [11]	図 17
3	clock	clockX > clock	図 18
3	clock	clockX < clock	図 20
4	master_clock	master_clock	図 21
4	clock	clock	図 17

1. 入力はデバイスピンによって駆動され、内部で Sync コンポーネントにより同期されます。Sync コンポーネントは、Component が使用するクロックと異なる内部クロックを使用してクロックを供給されます(すべての内部クロックは master_clock から派生)。

   このような方法で構成された入力の特性を解析する際は、clockX は Component のクロックより速い、同じ、遅い場合があります。master_clock と同じ場合もあります。これにより、図 12、図 13、図 15、および 図 16 に示す特性解析パラメーターが生成されます。

2. 入力はデバイスピンによって駆動され、master_clock を使用して同期されます。

   このような方法で構成された入力の特性を解析する際は、master_clock は Component のクロックより速い場合と同じ場合があります(遅いことはありません)。これにより、図 13 および 図 16 に示す特性解析パラメータが生成されます。

図 12. 入力構成 1 および 2、Sync Clock Frequency = Component Clock Frequency (clock と clockX のエッジアライメントは保証されません)

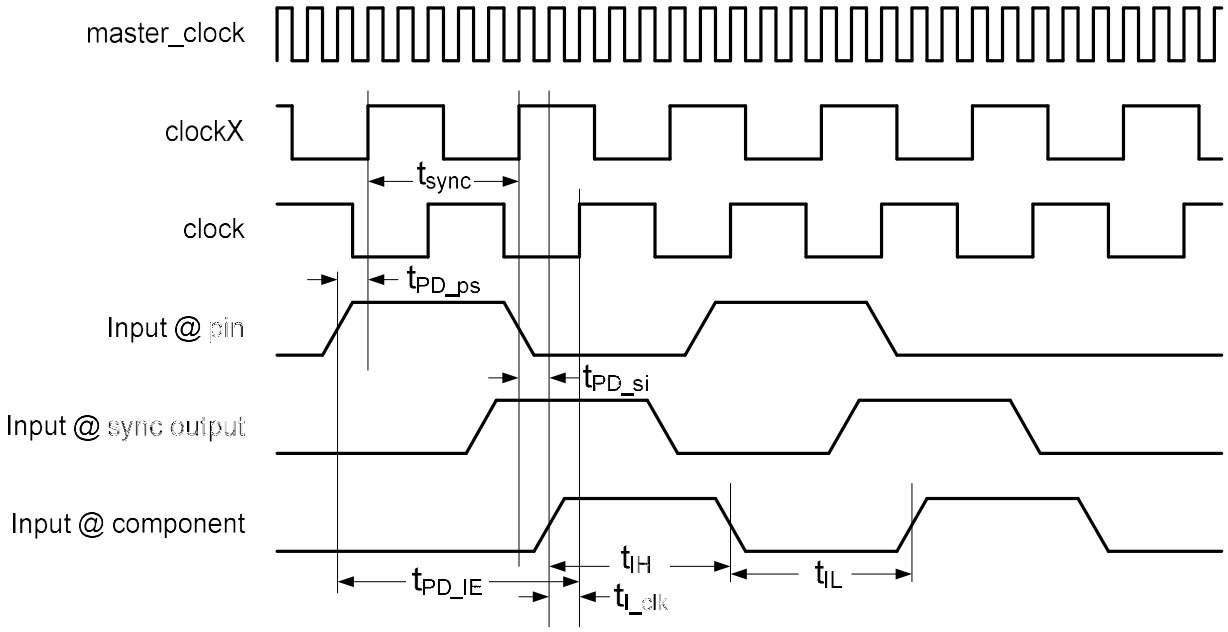


図 13. 入力構成 1 および 2、Sync Clock Frequency > Component Clock Frequency

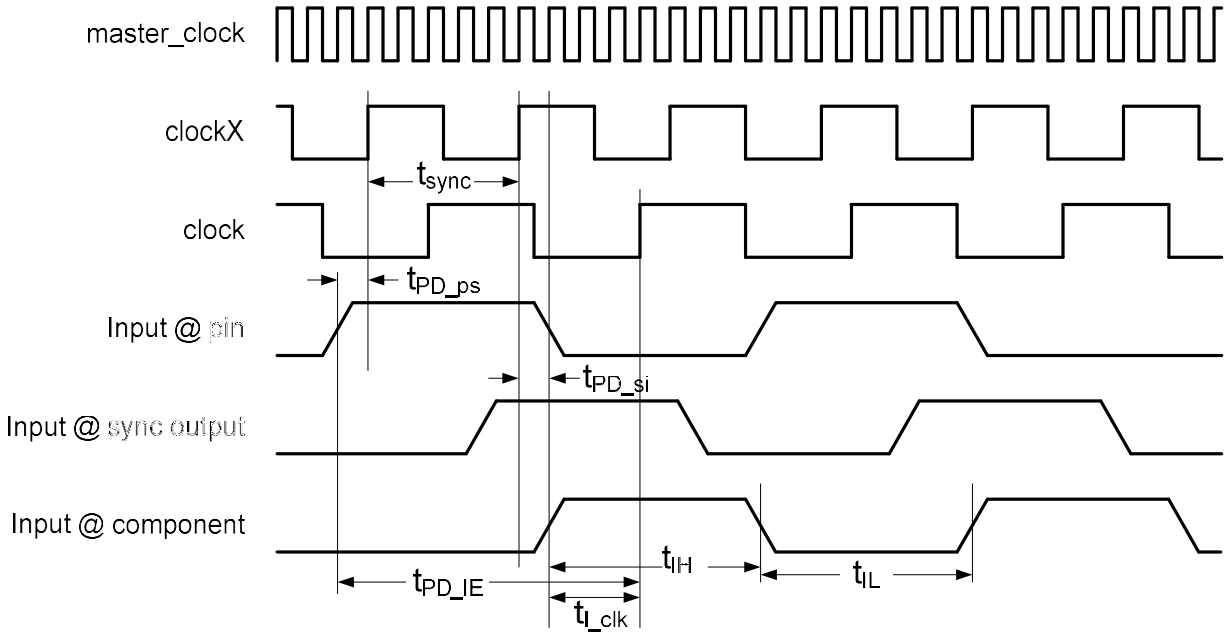


図 14. 入力構成 1 および 2、[Sync Clock Frequency == master_clock] > Component Clock Frequency

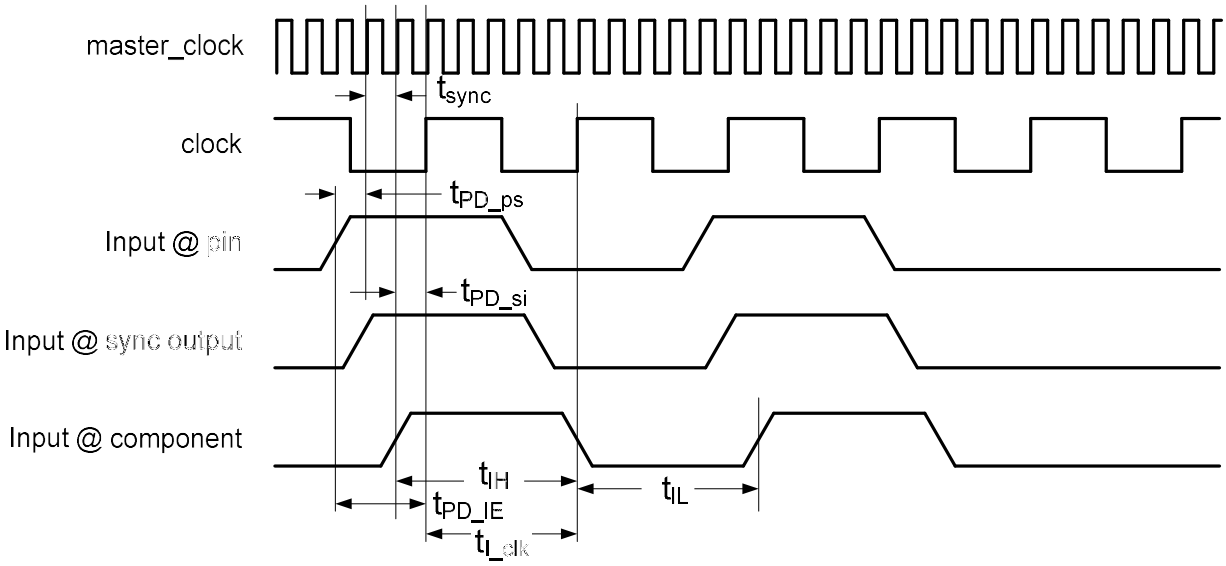


図 15. 入力構成 1、Sync Clock Frequency < Component Clock Frequency

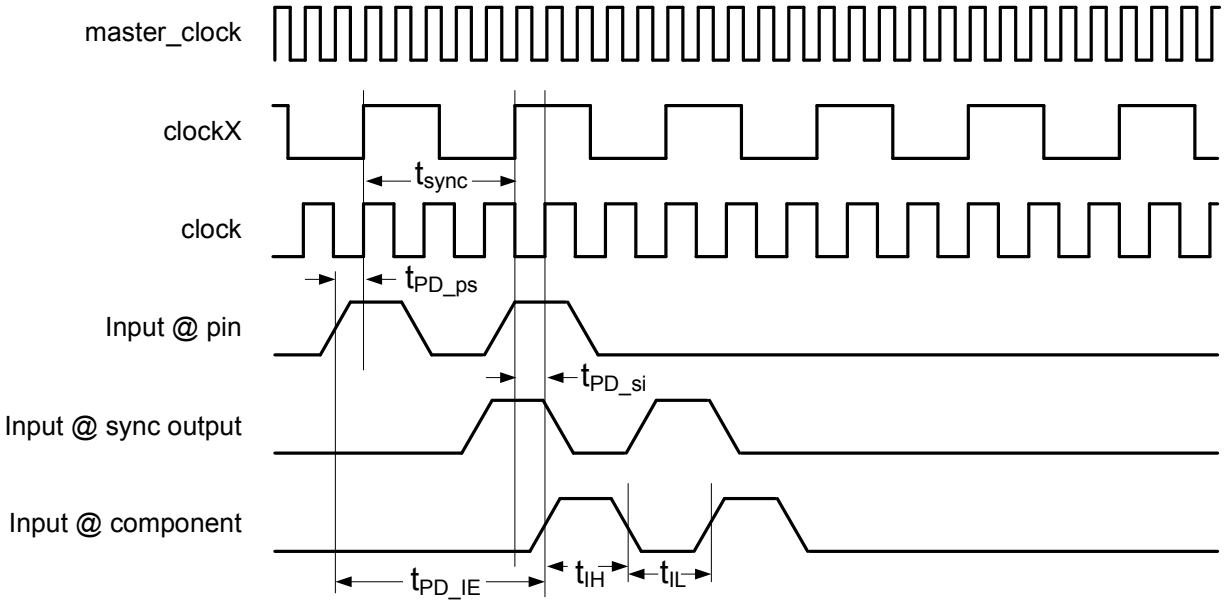
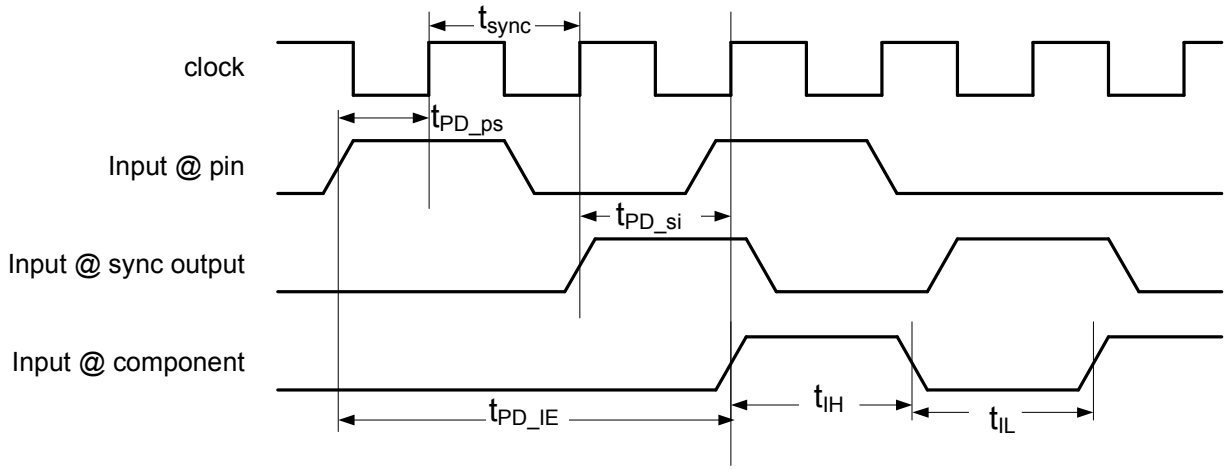


図 16. 入力構成 1 および 2、Sync Clock = Component Clock = master_clock



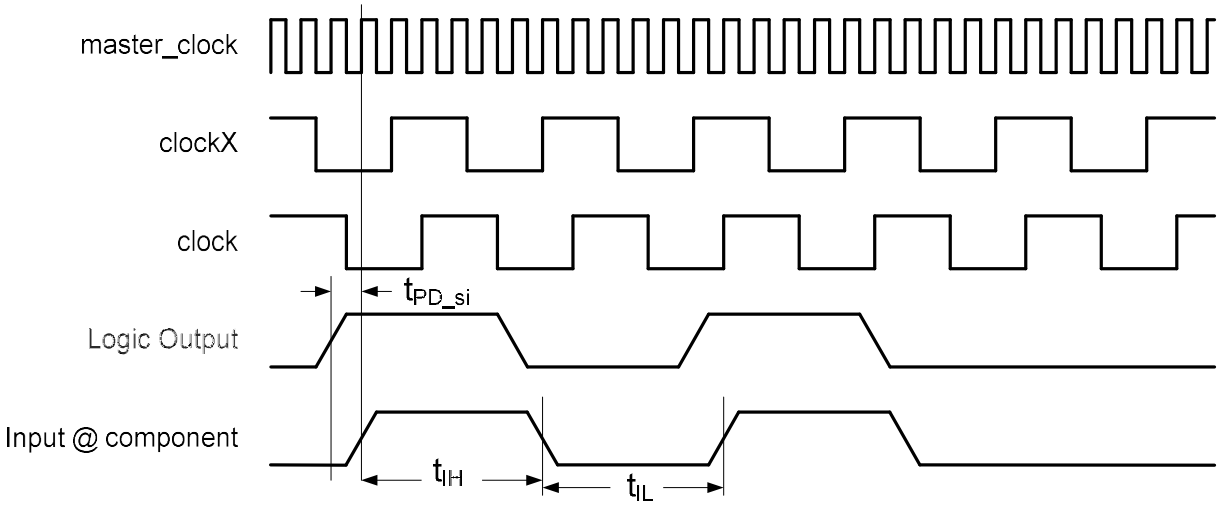
3. 入力は PSoC 内部の Logic に駆動されます。これは Component が使用するクロックとは異なるクロックをベースにして同期しています(すべての内部クロックは master_clock から派生)。

   このような方法で構成された入力の特性を解析する際は、シンクロナイザ クロックは、Component のクロックより速い、遅い、同じ場合があります。これにより、図 17、図 18、および 図 20 に示す特性解析パラメーターが生成されます。

4. 入力は PSoC 内部の Logic に駆動されます。これは Component が使用するクロックと同じクロックをベースにして同期しています。

   このような方法で構成された入力の特性を解析する際は、シンクロナイザ クロックは、Component のクロックと同じです。これにより、図 21 に示す特性解析パラメータが生成されます。

図 17. 入力構成 3 のみ、Sync Clock Frequency = Component Clock Frequency (clock と clockX のエッジアライメントは保証されません)



この図は、静的タイミング解析(STA)でのクロックについての情報を表しています。デジタルクロック領域のすべてのクロックは master_clock と同期します。但し、同じ周波数を持つ 2 つのクロックは、立ち上がりエッジのタイミングが一致しないことがあります。そのため、静的タイミング解析ツールは、クロックが同期しているエッジがどちらか判別できず、最低 1 master_clock サイクルを想定します。つまり t_{PD_si} はシステムの master_clock を制限する効果があります。この配線の遅延が大きいと、master_clock セットアップ時間に違反します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 18. 入力構成 3、Sync Clock Frequency > Component Clock Frequency

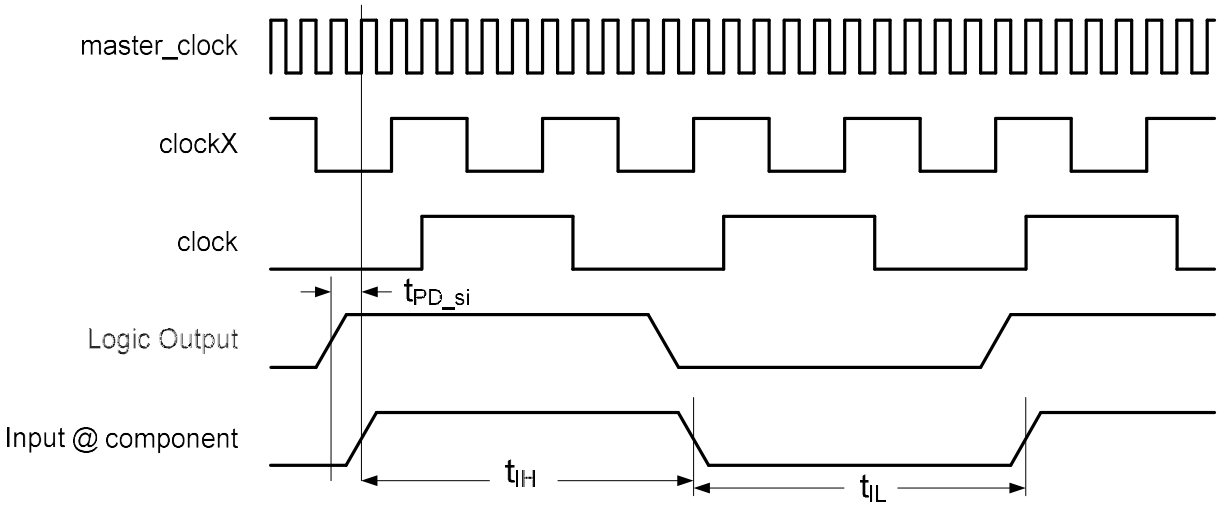


図 17 に示すのと類似した方法で、すべてのクロックは master_clock から派生します。STA は 1 master_clock サイクルに関する t_{PD_si} の制限を示します。この配線の遅延が大きいと、master_clock セットアップ時間に違反します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 19. 入力構成 3、Synchronizer Clock Frequency = master_clock > Component Clock Frequency

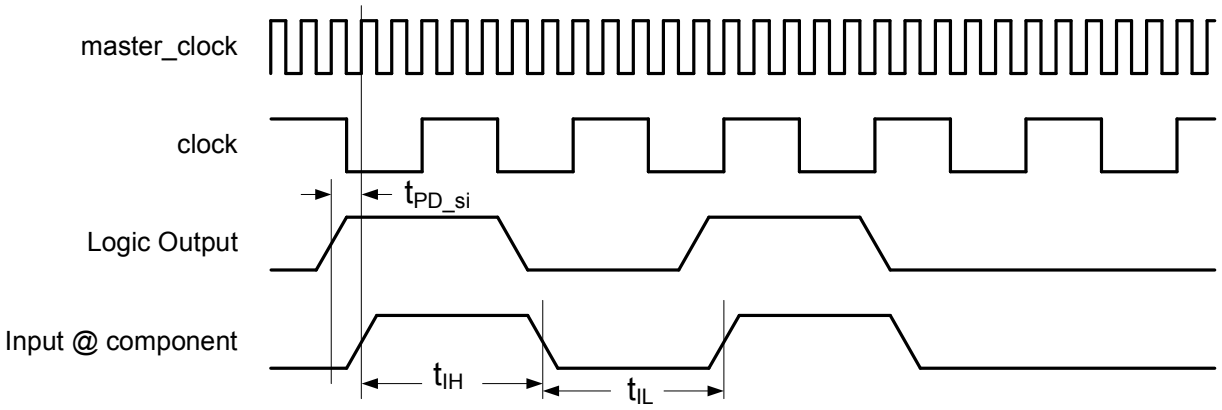


図 20. 入力構成 3、Synchronizer Clock Frequency < Component Clock Frequency

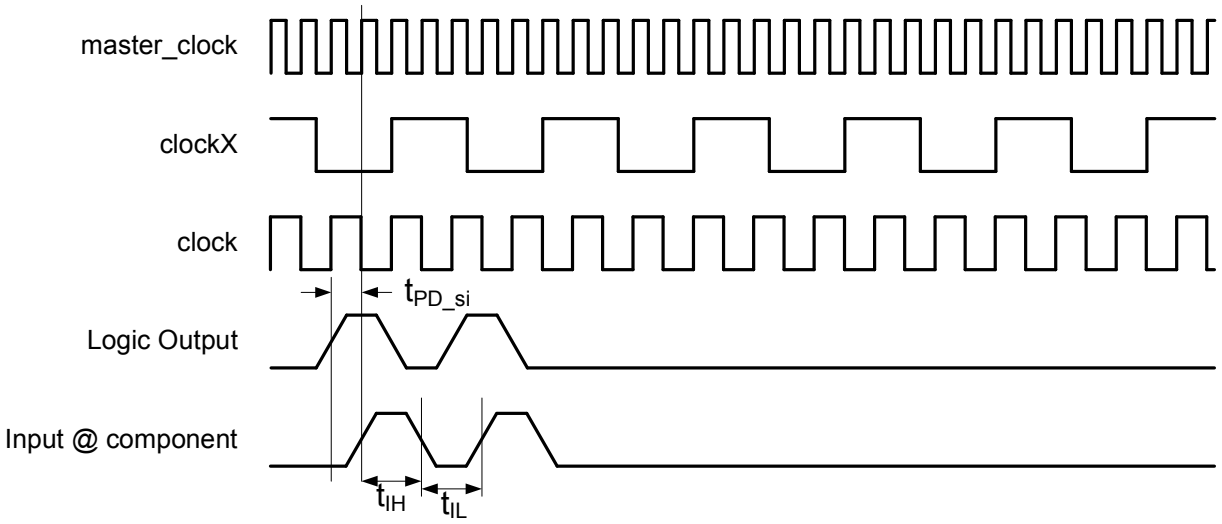
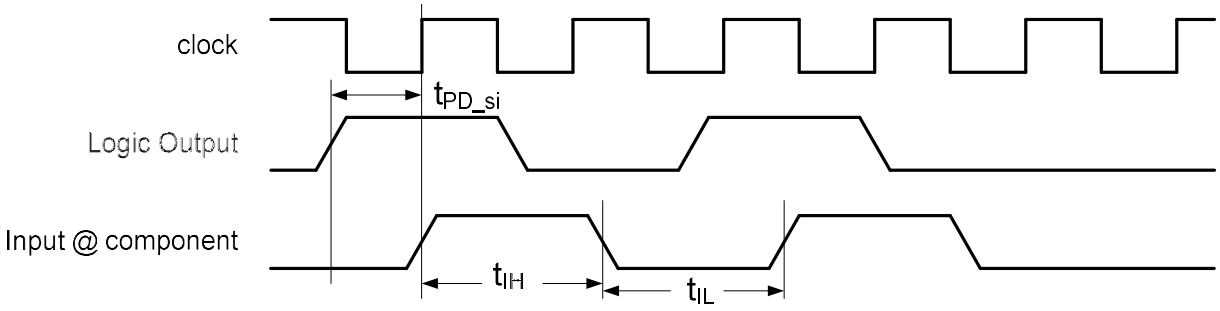


図 17 に示すのと類似した方法で、すべてのクロックは master_clock から派生します。STA は 1 master_clock サイクルに関する t_{PD_si} の制限を示します。この配線の遅延が大きいと、master_clock セットアップ時間に違反します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 21. 入力構成 4 のみ、Synchronizer Clock = Component Clock



このセクションでこれまで示した図の中で、実装を理解するために最も重要なパラメータは、 f_{CLOCK} と t_{PD_IE} です。t_{PD_IE} は t_{PD_ps} と t_{SYNC} (構成 1 と 2 のみ)、t_{PD_si}, 、t_{I_Clk} で定義されます。非常に重要な事は、t_{PD_si} が最大コンポーネントクロック周波数を定義するということです。t_{I_Clk} は STA の結果によるものではありませんが、t_{PD_IE} が登録された場合は重要となります。これはシンクロナイザと Component クロックの間の配線にあるマージンです。

t_{PD_ps} および t_{PD_si} は STA の結果に含まれています。

t_{PD_ps} を見つけるには、*_timing.html* ファイルで定義されている入力セットアップ時間を参照してください。この入力の Fan-out は複数の可能性があり、これらの配線の最大値を評価する必要があります。

-Setup times

-Setup times to clock BUS_CLK

Start	Register	Clock	Delay (ns)
input1(0):iocell.pad_in	input1(0):iocell.ind	BUS_CLK	16.500

t_{PD_si} は、Register-to-register times に定義されています。*_timing.html* ファイルを使用するには、ネット名を知っていなければなりません。このパスの Fan-out は複数の可能性があり、これらの配線の最大値を評価する必要があります。

-Register-to-register times

-Destination clock clock

Destination clock clock (Actual freq: 24.000 MHz)

+Source clock clock

-Source clock clock_1

Source clock clock_1 (Actual freq: 24.000 MHz)
Affected clock: BUS_CLK (Actual freq: 24.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
\Sync_1:genblk1[0]:INST\:synccell.syncq	\PWM_1:PWMUDB:runmode_enable\:macrocell.mc_d	7.843	127.508 MHz	24.000 MHz	

出力配線遅延

出力の配線遅延の特性解析を行う場合、STA の結果の中からデータを見つけるために、信号の出力先を考慮しなければなりません。このコンポーネントでは、すべての出力が Component クロックに同期されています。出力は 2 つのカテゴリのうち、いずれかに該当します。出力は、デバイス内の別のコンポーネントへ送られるか、デバイス外のピンに進むかのどちらかです。前者の場合、上述の Logic-to-input descriptions に記載されている Register-to-register times を見ます(ソースクロックは Component クロックです)。後者の場合、*_timing.html* STA の結果の Clock-to-Output times を見ます。

コンポーネントの更新履歴

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	変更内容	変更理由 / 影響
2.20	UDB 実装での Verilog 変更	ハードウェアイネーブル信号が使われている場合、TC 出力は特定の条件下でミスされる場合を修正します
	割り込み信号はPSoC 5 FF 実装で利用できないということを記録します	シリコンではサポートされないため、この機能は削除されました
	Cancel ボタンを常に使用できるように、カスタマイザを更新しました	一部のエラー状況下では、Cancel ボタンは利用できませんでした
	詳細なデータシートアップデート	タイマの実装は各実装毎 (UDB、PSoC 3 FF、PSoC 5 FF) に異なっていますが、これらの違いについて適切に説明していませんでした。特に。「機能の詳細」の設定セクションにある波形を参照してください。
2.10	Verilog とカスタマイザ関連のアップデート	トリガロジックと GUI 関連事項の課題を修正
	「Capture Mode」が「None」に設定されている場合は、"Interrupt on Capture" が利用できません	「Capture Mode」が「None」に設定されていて、利用可能にすべきでない場合でも、"Interrupt on Capture" チェックボックスオプションが利用できました
2.0	入力の同期	専用機能ブロック実装で、ブロック入力ですべての入力を同期。
	関数Timer_GetInterruptSource() をマクロに変更	関数Timer_GetInterruptSource() は関数 Timer_ReadStatusRegister() と全く同じ実装。コードスペースを節約するため、これを関数Timer_ReadStatusRegister() のマクロ代替に変換。
	出力がコンポーネントクロックに登録されるように変更	コンポーネントの出力上のグリッチを避けるために、すべての出力が同期されることが必要だった。これは、リソース使用量を抑えるためにできる限りデータパスの内部で行われます。
	補助コントロールレジスタに書き込む際の、実装された重要領域	補助コントロールレジスタに書き込む際は、関数 CyEnterCriticalSectionとCyExitCriticalSectionsを使用。他のプロセススレッドによる変更を避けるため。

バージョン	変更内容	変更理由 / 影響
	SetCaptureMode() APIを使用してキャプチャモードを設定する際の不適切なマスキングを修正。	キャプチャモードの設定に使用するマスキングが誤った値であった。
	データシートに特性データを追加	
	データシートのマイナーな編集と更新	

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。