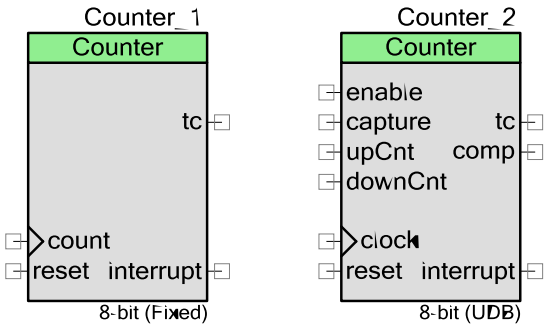
カウンタ
2.0

特長



- 固定機能 (FF) とユニバーサル デジタル ブロックによる (UDB) の実装
- 8、16、24、または 32 ビット カウンタ
- アップ、ダウン、またはアップダウン設定
- 比較出力（オプション）
- キャプチャ入力（オプション）
- 他のコンポーネントと同期するためのイネーブルおよびリセット入力
- 連続またはワンショット実行モード

概要説明

このカウンタ コンポーネントは、イベントをカウントする手段を提供します。カウンタの基本機能を実装できるほか、キャプチャ、出力比較、カウント方向制御といった高度な機能も提供します。

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。デバイス内での FF リソースの詳細については、デバイスのデータシートまたはテクニカル リファレンス マニュアルを参照してください。

機能	FF	UDB
ビット数	8 または 16	8、16、24、または 32
Run Mode (実行モード)	連続またはワンショット	連続またはワンショット
カウンタ モード	ダウンのみ	アップ、ダウン、またはアップダウン
イネーブル入力	X (ソフトウェア イネーブルのみ)	○ (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	X	○

機能	FF	UDB
キャプチャ モード	なし	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	X (キャプチャ レジスタ 1 個)	○ (キャプチャ レジスタ 4 個まで)
リセット入力	○	○
カウント終了出力	○	○
比較出力	X	○
比較 モード	なし	<、 ≤、=、 ≥、 >、またはソフトウェア制御
割り込み出力	○	○
割り込み条件	TC	TC、キャプチャ、比較
期間レジスタ	○	○
期間リロード	○ (リセットまたは TC 時に常にリロード)	○ (リセット、TC、キャプチャ、比較時にリロード)
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
サンプリング クロック入力	X	コンポーネントのサンプリング入力信号用に明示的なクロック信号（コンポーネント クロック）が必要

カウンタの使用

カウンタはデフォルトでカウント入力のエッジ イベント数のカウントに使用されますが、他にも次のような使用方法があります。

- クロック デバイダ: クロックをカウント入力に分周し、比較またはカウント終了出力を分周クロック出力として使用する。
- 周波数カウンタ: 既知の期間を持つ信号をカウンタのイネーブル入力に接続し、カウント入力上で被測定信号をカウントする。
- 直交デコーダの出力など余事象の測定ツール。

注 タイマ コンポーネントは、イベント間の時間の測定に重点が置かれるような状況に適しています。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。

入出力接続

ここでは、このカウンタ コンポーネントの各種の入出力接続について説明します。一部の入出力の記号は、その入出力の説明に表示されている条件の下に隠されている場合があります。

注 特記がない限り、すべての信号はアクティブ HIGH です。カウンタへの入力はすべてカウンタ外で同期する必要があります。

入力	隠し表示	説明
clock/ count	N	固定機能実装では、クロック入力はカウントする信号を定義します。カウンタは、クロック入力の各立ち上がりエッジでデクリメントされます。 UDB 実装では、4 つのモードにより、入力クロックとカウントが隠されるか表示されるかが決まります。 • アップ カウンタ/ダウン カウンタ: カウント入力はカウントする信号を定義し、クロック入力ごとにサンプルされます。カウンタは、カウント入力の各立ち上がりエッジでアップまたはダウンをカウントします (**クロック モード** パラメータによって異なる)。カウンタはクロック入力に同期してカウントします。 • Clock + upCnt & dwnCnt: upCnt と dwnCnt ターミナルはカウントの方向、クロック入力に対してカウンタがインクリメントされるかデクリメントされるか、または NOP かを示します。このモードではカウント入力は不可視です。 クロックと方向: クロック入力はカウントする信号を定義します。方向入力は、カウントアップかカウントダウンかというカウンタの方向を定義します。カウンタはクロック入力に同期し、立ち上がりエッジでカウントします。最大で、カウント入力はクロック入力の周波数の半分までにできます。
upCnt	Y	カウンタへの信号をインクリメントします。**クロック モード** が「Clock + upCnt & dwnCnt」の場合、この入力を dwnCnt 入力およびクロック入力といっしょに使用することでカウンタをエンコーダとして使用できるようになります。この入力の立ち上がりエッジで、カウント値がインクリメントされます。
dwnCnt	Y	カウンタへの信号をデクリメントします。カウンタの**クロック モード**が「Clock + upCnt & dwnCnt」に設定されている場合、この入力を upCnt 入力およびクロック入力といっしょに使用して、カウンタをエンコーダとして使用できます。この入力の立ち上がりエッジで、カウント値がデクリメントされます。
up_ndown	Y	カウンタのカウント方向を定義します。この入力は、**クロック モード** が方向制御を有効にする (「Count + Direction Inputs」)に設定されている場合にのみ使用可能です。カウント入力の立ち上がりエッジでこの入力が 1 ならカウンタがインクリメントされ、0 ならデクリメントされます。

入力	隠し表示	説明
reset	N	reset 入力は、カウンタを開始値にリセットします。 • 「アップ カウンタ」設定では、開始値はゼロです。 • 「Down Counter」 (ダウン カウンタ)、「Count Input and Direction」 (カウント入力と方向)、「Clock With UpCnt & DwnCnt」 (クロックと UpCnt & DwnCnt) の設定では、開始値は期間レジスタの現在値に設定されます。 reset 入力は count/clock 入力時にサンプルされます。 **注** PSoC 3 ES2 シリコンでは、固定関数カウンタのカウント終了 ピンはリセットで HIGH に保たれます。この回路図の修正は、このデータシートの機能説明セクションの「固定機能ブロックのリセット」を参照してください。 PSoC 3 ES3 以降のシリコンでは、固定機能カウンタのカウント終了 ピンはリセットで LOW に保たれます。reset 入力は、PSoC3 ES3 と PSOC5 ES1 UDB カウンタ実装での制御レジスタのリセット、さらにはワンショット実行モード機能の実装に使用されます。
enable	Y	カウンタのハードウェア イネーブル。この入力は、**イネーブル モード** パラメータがハードウェアに設定されているときに可視となります。
capture	Y	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、**キャプチャ モード** パラメータが「なし」以外のモードに設定されているときに可視となります。キャプチャは、キャプチャ モードの設定に従い、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この入力に適用されます。 capture 入力は clock 入力のタイミングでサンプルされます。

出力	隠し表示	説明
tc	N	Terminal Countは、カウント値がTerminal Countに等しいことを示す同期出力です。出力は、カウンタのクロック入力に同期されます。信号は、カウント値がカウント終了に一致した後 1 クロック サイクルで HIGH になり、カウント値がTerminal Countに等しい間 HIGH のままになります。ダウン カウンタとして実装した場合、カウント終了値はゼロです。アップ カウンタとして実装した場合、この値は最大カウント (期間) になります。カウンタは、カウントがカウント終了になったときに無効になり、再度有効にされるまで HIGH のままになります。
comp	Y	compare 出力は、**比較 モード** パラメータの設定に基づいて比較値に比較したカウント値を示します。出力は、比較 イベント後のクロック入力 1 クロック サイクルで HIGH になります。
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 OR をとって、最終出力信号が作成されます。割り込みのソースは 「Compare」、「Terminal Count」、または「Capture」です。

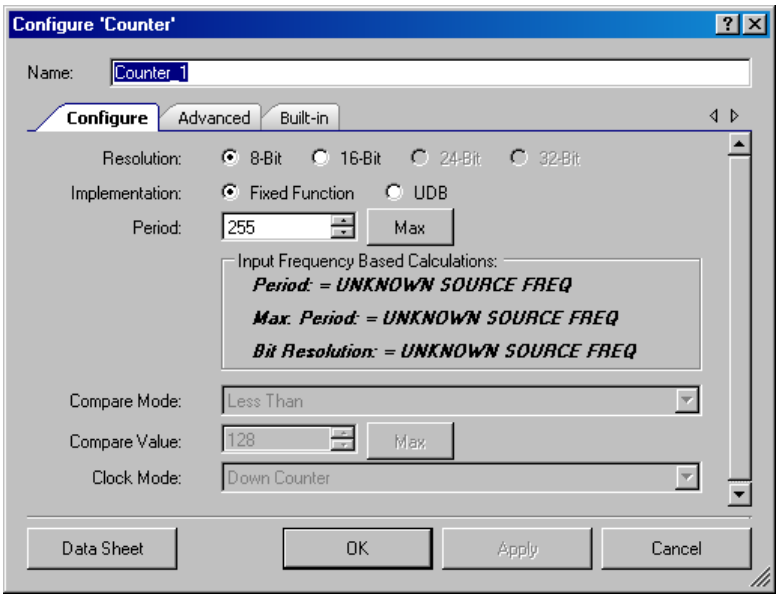
パラメータおよびセットアップ

カウンタを設計にドラッグ&ドロップして組み込み、ダブルクリックすると、[Configure] (設定) ダイアログが開きます。

ハードウェアとソフトウェア設定オプション

ハードウェア設定オプションにより、プロジェクトが合成されハードウェアに配置される方法が変わります。これらのオプションのどれかを変更した場合は、ハードウェアを再構成する必要があります。ソフトウェア設定オプションは合成や配置に影響しません。ビルド前にこれらのパラメータを設定する際、それらの初期値を設定することになりますが、これらの値は後でいつでも提供されている API を使って変更できます。次のセクションで説明するほとんどのパラメータは、ハードウェア オプションです。ソフトウェア オプションについてはあまり説明されていません。

Configure (設定) タブ



Resolution (分解能)

Resolution パラメータは、カウンタ コンポーネントのビット幅分解能を定義します。この値は最大カウント値 255、65535、16777215、4294967295 に対して、それぞれ 8、16、24、32 に設定できます。

Implementation (実装)

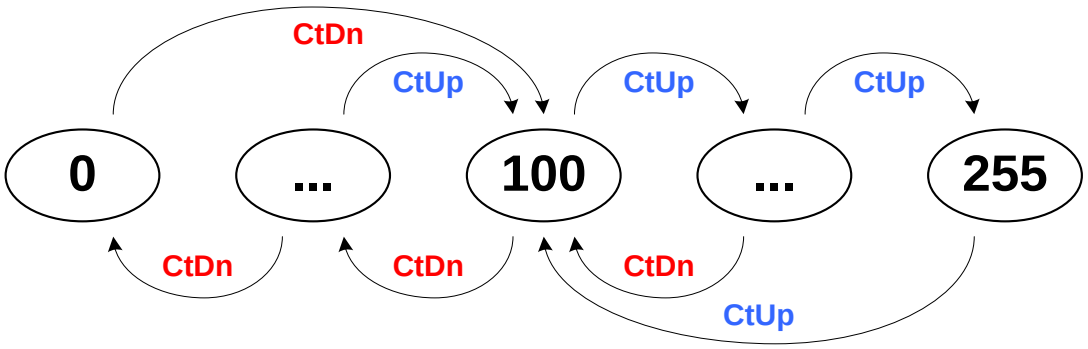
Implementation パラメータを使うと、カウンタを固定機能ブロックで実装するか、または UDB で実装するかを選択できます。FF を選択すると、UDB 関数は無効になります。

Period (期間)(ソフトウェア オプション)

Period パラメータは、カウンタ 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) を定義します。このパラメータは、期間レジスタにロードする初期値を定義します。この値は、Counter_WritePeriod() API を使ってソフトウェアでいつでも変更できます。

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** パラメータに対し、**期間** 値の最大値はそれぞれ (2^8)-1、(2^16)-1、(2^24)-1、(2^32)-1、255、65535、16777215、4294967295 と定義されています。

Clock Mode (クロック モード) を「Clock with UpCnt & DwnCnt] (クロックと UpCnt & DwnCnt)または「Count Input + Direction] (カウント入力 + 方向)として設定すると、開始時とカウンタがオーバーフローまたはアンダーフローしたときにはいつでもカウンタは期間に設定されます。これらのクロック モードでは、期間値はオール 1 またはオール 0 に設定しないでください。通常は、期間範囲の中点の期間値を保持します (8 ビット カウンタでは 0x7F)。下図に、**Clock Mode** を「Count Input + Direction」に設定した例を示します。



Compare Mode (比較 モード)(ソフトウェア オプション)

Compare Mode パラメータは、comp 出力信号の動作を設定します。この信号は、比較値パラメータと現在のカウンタ値との比較状態を示します。このパラメータは初期設定を定義します。この値は、カウンタ コンポーネントの比較動作を再設定することでいつでも更新できます。

Compare Mode は、以下のいずれかの値に設定できます。

- Less Than (より小) – カウンタ値が比較値より小
- Less Than Or Equal To (以下) – カウンタ値が比較値以下

- Equal To (等しい) – カウンタ値が比較値と等しい
- Greater Than (より大) – カウンタ値が比較値より大
- Greater Than Or Equal To (以上) – カウンタ値が比較値以上
- Software Controlled (ソフトウェア制御) – 比較モードは実行時に Counter_SetCompareMode() API で上記の 5 つの比較モードのいずれかを呼び出して設定できます。

Compare Value (比較値)(ソフトウェア オプション)

Compare Value パラメータは、カウンタの比較レジスタにロードする初期値を定義します。この値は、比較出力の動作を定義するために選択した Compare Mode パラメータと併用します。

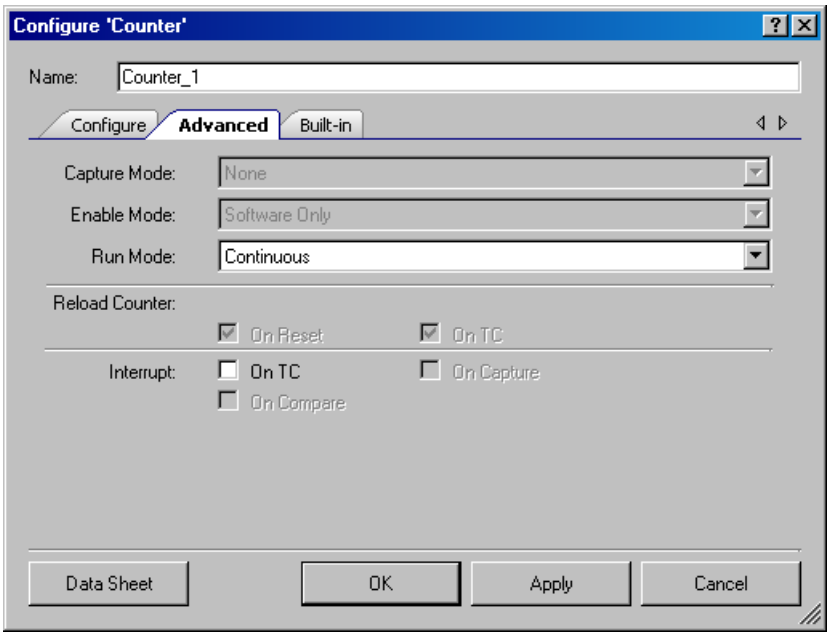
この値には、0 ～ (2^分解能-1) の任意の符号なし整数を指定できますが、期間値以下でなければなりません。

Clock Mode

Clock Mode パラメータは、カウンタのカウント方法を設定します。このモードは、以下のいずれかの値に設定できます。

- Count Input + Direction (カウント入力 + 方向) – カウンタが双方向性カウンタであり、入力クロックの各立ち上がりエッジで up_ndown 入力が HIGH の間カウントアップし、入力クロックの各立ち上がりエッジで up_ndown が LOW の間カウントダウンします。
- Clock with UpCnt & DwnCnt (クロック + UpCnt & DwnCnt) – カウンタが双方向性カウンタであり、クロック入力に対する upCnt 入力の各立ち上がりエッジでカウンタをインクリメントし、クロック入力に対する dwnCnt 入力の各立ち上がりエッジでカウンタをデクリメントします。このモードでは、クロック入力周波数は UpCnt および DwnCnt 入力の周波数の少なくとも 2 倍でなければなりません。
- Up Counter (アップ カウンタ) – カウンタがアップ カウンタのみです。これは、カウンタが有効になっている間、クロック信号に対するカウント入力の立ち上がりエッジでインクリメントするように設定します。
- Down Counter (ダウン カウンタ) – カウンタがダウン カウンタのみです。これは、カウンタが有効になっている間、クロック信号に対するカウント入力の立ち上がりエッジでデクリメントするように設定します。

Advanced (詳細) タブ



Capture Mode (キャプチャ モード)

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、以下のいずれかの値に設定できます。

- None (なし) – 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 (立ち上がりエッジ) – 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タ値をキャプチャします。
- Falling Edge (立ち下がりエッジ) – 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タ値をキャプチャします。
- Either Edge (両エッジ) – クロック入力に対するキャプチャ入力の立ち上がりと立ち上がりエッジでカウンタ値をキャプチャします。
- Software Controlled (ソフトウェア制御) – Software Controlled モードでは、*Counter.h* ヘッダ ファイルで定義されている列挙型キャプチャ モード タイプで制御レジスタ Counter_CTRL_CAPMODE_MASK のキャプチャ モード ビットを設定することで実行時にモードを設定します。

Enable Mode (イネーブル モード)

Enable Mode パラメータは、カウンタ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 (ソフトウェア) – カウンタは、制御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 (ハードウェア) – カウンタは、イネーブル ビットのみに基づいて有効になります。
- Software and Hardware (ソフトウェアとハードウェア) – カウンタは、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 (実行モード)

Run Mode パラメータを使うと、カウンタ コンポーネントを連続実行するかワンショット モードで実行するかを設定できます。

- Continuous (連続) – カウンタはイネーブルになっている間、連続実行します。
- One Shot (ワンショット) – カウンタは 1 期間実行し、カウント終了で終了します。リセット後、別の 1 サイクルを開始します。UDB カウンタの場合は停止時に期間をカウント レジスタにリロードし、固定機能カウンタの場合はカウント レジスタはカウント終了時にも残ります。

Reload Counter (カウンタのリロード)

Reload Counter パラメータを使うと、カウンタ値をリロードするタイミングを設定できます。カウンタ値は以下のうち選択したイベントが 1 つ以上発生したときにリロードされます。カウンタに開始値がリロードされます (アップ カウンタではゼロ値、ダウン カウンタでは最大カウントまたは期間値)。

- On Capture (キャプチャ時) – カウンタ値はキャプチャ イベントが発生したときにリロードされます。このパラメータはデフォルトで「偽」に設定されています。このパラメータは実装で「UDB」が選択されたときにのみ表示されます。
- On Compare (比較時) – カウンタ値は比較が「真」の イベントが発生したときにリロードされます。このパラメータはデフォルトで「偽」に設定されています。このパラメータは実装で「UDB」が選択されたときにのみ表示されます。
- On Reset (リセット時) – カウンタ値はリセット イベントが発生したときにリロードされます。このパラメータはデフォルトで「真」に設定されています。このパラメータは常に表示されています。固定機能カウンタの場合、これは変更できません。UDB の場合はオフにできます。

- On TC (TC 時) – カウンタ値は、カウンタがオーバーフロー (アップ カウント モード) またはアンダーフロー (ダウン カウント モード) したときにリロードされます。このパラメータはデフォルトで「真」に設定されています。固定機能カウンタの場合、これは変更できません。UDB の場合はオフにできます。

  クロック モードが「Clock with UpCnt & DwnCnt」に設定されている場合、このオプションはカウンタが 0x00 またはオール 0xFF のときに期間値をリロードします。

Interrupt (割り込み)

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 1 つ以上発生したときにリロードされます。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 (TC 時) – このパラメータは常にアクティブで、デフォルト設定は「偽」です。
- On Capture (キャプチャ時)– このパラメータはデフォルトで「偽」に設定されています。これは常に表示されていますが、**Implementation** パラメータが「UDB」に設定されているときのみアクティブになります。
- On Compare (比較時)– このパラメータはデフォルトで「偽」に設定されています。これは常に表示されていますが、**Implementation** パラメータが「UDB」に設定されているときのみアクティブになります。

クロック選択

カウンタ コンポーネントでは、

- **Clock Mode** パラメータが「Up Counter」または「Down Counter」に設定されていると、立ち上がりエッジがカウンタとなる任意の信号がカウント入力となり得ます。コンポーネントへのクロック入力がカウント入力をサンプルするために使用され、カウント入力の周波数の少なくとも 2 倍の周波数を持たなければなりません。
- **Clock Mode** パラメータが「Count input with Direction」に設定されていると、立ち上がりエッジがカウンタとなる任意の信号がカウント入力となり得ます。カウンタは、up_ndown の入力値に応じてカウントアップまたはカウントダウンされます。
- **Clock Mode** パラメータが「Clock with upCnt & dwnCnt」に設定されていると、クロック入力に対して upCnt と dwnCnt の立ち上がりエッジがサンプルされます。カウンタは、upCnt 信号の立ち上がりエッジでカウントされ、dwnCnt 信号の立ち上がりエッジでカウントダウンされます。

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。

固定機能コンポーネント

FF ブロックを使用するように設定されている場合、カウンタ コンポーネントには以下の制限が課せられます。

- カウント入力はクロック 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 (設定) ダイアログを開いて、**Clock Type (クロック タイプ)** パラメータを「Existing」 (既存)として設定し、**Source (ソース)** パラメータを「BUS_CLK 」(バス クロック)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 コンポーネントでは

どのソースからのどのデジタル信号をもカウント/クロック入力に接続できます。その信号の周波数は、このデータシートの仕様セクションで定義されている周波数範囲に制限されます。どのカウンタ クロック モードの場合も、カウント入力は最大でもクロック入力周波数の半分でなければなりません。

配置

PSoC Creator は、**Implementation** パラメータに基づいてカウンタ コンポーネントをデバイス内に配置します。「Fixed Function」 (固定機能)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 アレイの周辺に配置されます。

リソース

分解能	デジタル ブロック					API メモリ (バイト数)		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	制御レジスタ	Counter7	Flash	RAM	
8 ビット UDB 基本アップ/ダウン カウンタ	1	6	1	1	0	240	4	0
8 ビット UDB カウンタと方向 *	1	10	1	1	0	240	4	0

分解能	デジタル ブロック					API メモリ (バイト数)		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	制御レジスタ	Counter7	Flash	RAM	
8 ビット FF カウンタ	0	0	0	0	0	352	5	0
16 ビット UDB 基本アップ/ダウン カウンタ	2	6	1	1	0	298	5	0
8 ビット UDB カウンタと方向 *	2	14	1	1	0	452	6	0
16 ビット FF カウンタ	0	0	0	0	0	372	6	0
24 ビット基本カウンタ	3	6	1	1	0	448	8	0
32 ビット 基本アップ/ダウン カウンタ	4	6	1	1	0	448	8	0

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。

PSoC Creator はデフォルトで、設計内のコンポーネントの最初のインスタンスに「Counter_1」というインスタンス名を割り当てます。そのインスタンス名は、識別子の構文ルールに従う任意の固有名に変更できます。インスタンス名は、各グローバル関数名、変数、定数記号のプリフィックスとなります。便宜上、次の表ではインスタンス名として「Counter」を使っています。

関数	説明
Counter_Start()	initVar 変数を設定し、Counter_Init() 関数を呼び出して、Enable 関数を呼び出します。
Counter_Stop()	カウンタを無効にします。
Counter_SetInterruptMode()	割り込み出力のソースを有効または無効にします。
Counter_ReadStatusRegister()	ステータス レジスタの現在の状態を返します。

関数	説明
Counter_ReadControlRegister()	制御レジスタの現在の状態を返します。
Counter_WriteControlRegister()	制御レジスタのビット フィールドを設定します。
Counter_WriteCounter()	新しい値を直接カウンタ レジスタに書き込みます。
Counter_ReadCounter()	キャプチャを強制し、キャプチャ値を返します。
Counter_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。
Counter_WritePeriod()	期間レジスタに書き込みます。
Counter_ReadPeriod()	期間レジスタを読み取ります。
Counter_WriteCompare()	比較レジスタに書き込みます。
Counter_ReadCompare()	比較レジスタを読み取ります。
Counter_SetCompareMode()	比較モードを設定します。
Counter_SetCaptureMode()	キャプチャ モードを設定します。
Counter_ClearFIFO()	キャプチャ FIFO をクリアします。
Counter_Sleep()	カウンタを停止し、ユーザ設定を保存します。
Counter_Wakeup()	ユーザ設定を復元し、有効にします。
Counter_Init()	Configure (設定) ダイアログでの設定に従ってカウンタを初期化または復元します。
Counter_Enable()	カウンタを有効にします。
Counter_SaveConfig()	カウンタ設定を保存します。
Counter_RestoreConfig()	カウンタ設定を復元します。

グローバル変数

変数	説明
Counter_initVar	カウンタが初期化されているかを示します 変数は 0 に初期化され、Counter_Start() が最初に呼び出されたときに 1 に設定されます。これにより、最初に Counter_Start() ルーチンを呼び出した後で再初期化することなく再起動できるようになります。 コンポーネントの再初期化が必要な場合には、Counter_Start() または Counter_Enable() 関数の前に Counter_Init() 関数を呼び出すことができます。

void Counter_Start(void)

説明 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。Counter_Start() は initVar 変数を設定し、Counter_Init() 関数を呼び出して、Counter_Enable() 関数を呼び出します。

パラメータ なし

戻り値 なし

副作用 initVar 変数がすでに設定されている場合は、この関数は Counter_Enable() 関数を呼び出すだけです。

void Counter_Stop(void)

説明 ソフトウェア イネーブル モードでのみカウンタを無効にします。

パラメータ なし

戻り値 なし

副作用 Enable Mode が「Hardware Only (ハードウェアのみ)」に設定されている場合は、この関数は無効です。

void Counter_SetInterruptMode (uint8 interruptSource)

説明 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「ステータス レジスタ」セクションを参照してください。

戻り値 なし

副作用 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 Counter_ReadStatusRegister (void)

説明 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

戻り値 uint8: 現在のステータス レジスタ値。ステータス レジスタ ビット：
[7] : 未使用 (0)
[6] : FIFO が空でない
[5] : FIFO が満杯
[4] : キャプチャ ステータス
[3] : アンダーフロー ステータス
[2] : オーバーフロー ステータス
[1] : A0 ゼロ ステータス
[0] : 比較出力。
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」セクションを参照してください。

副作用 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」セクションを参照してください。

uint8 Counter_ReadControlRegister(void)

説明 制御レジスタの現在の状態を返します。この関数は制御レジスタで定義したモードの 1 つが実際に使われている場合にのみ使用可能です。

パラメータ なし

戻り値 uint8: 現在の制御レジスタ値。制御レジスタ ビット：
[7] :Counter Enable
[6:5] : 未使用
[4:3] : Capture Mode select
[2:0] : Compare Mode select
ビット定義については、このデータシートの「制御 レジスタ」セクションを参照してください。

副作用 なし

void Counter_WriteControlRegister(uint8 control)

説明 制御レジスタのビット フィールドを設定します。この関数は制御レジスタで定義したモードの 1 つが実際に使われている場合にのみ使用可能です。

パラメータ uint8: 制御レジスタ ビットフィールド。制御レジスタ ビット :
[7] :Counter Enable
[6:5] :未使用
[4:3] : Capture Mode select
[2:0] : Compare Mode select
ビット定義については、このデータシートの「制御 レジスタ」セクションを参照してください。

戻り値 なし

副作用 なし

void Counter_WriteCounter (uint8/16/32 count)

説明 新しい値を直接カウンタ レジスタに書き込みます。

パラメータ (uint8/16/32) 新しいカウンタ値。24 ビット カウンタでは、このパラメータは uint32 です。

戻り値 なし

副作用 カウンタ値を上書きします。これは、比較出力、カウント終了出力、または期間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、関数の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カウンタを無効にしてください。

uint8/16/32 Counter_ReadCounter (void)

説明 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

戻り値 (uint8/16/32) カウンタの現在値、24 ビット カウンタでは、戻り値タイプは uint32 です。

副作用 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)。

uint8/16/32 Counter_ReadCapture (void)

説明 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)。

パラメータ なし

戻り値 (uint8/16/32) カウンタの現在値、24 ビット カウンタでは、戻り値タイプは uint32 です。

副作用 なし

void Counter_WritePeriod (uint8/16/32 period)

説明 期間レジスタに書き込みます。

パラメータ (uint8/16/32)新しい期間値。24 ビット カウンタでは、このパラメータは uint32 です。

戻り値 なし

副作用 カウンタ出力の期間はカウンタがリロードされるまで変化しません。

uint8/16/32 Counter_ReadPeriod (void)

説明 期間レジスタを読み取ります。

パラメータ なし

戻り値 (uint8/16/32)現在の期間値。24 ビット カウンタでは、戻り値タイプは uint32 です。

副作用 なし

void Counter_WriteCompare (uint8/16/32 compare)

説明 比較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) 新しい比較値。24 ビット カウンタでは、このパラメータは uint32 です。

戻り値 なし

副作用 比較出力は、カウンタに書き込まれた値と現在値によってはすぐに変更されることがあります。

uint8/16/32 Counter_ReadCompare (void)

説明 比較レジスタを読み取ります。この関数は UDB 実装でのみ使用可能です。

パラメータ なし

戻り値 (uint8/16/32) 24-bit カウンタでは、戻り値タイプは uint32 です。

副作用 なし

void Counter_SetCompareMode (uint8 compareMode)

説明 比較モードを設定します。この関数は UDB 実装でソフトウェア比較モードが選択された場合にのみ使用可能です。

パラメータ (uint8) 列挙型比較モード。「制御レジスタ」セクションも参照してください。
```
Counter__B_COUNTER__LESS_THAN
Counter__B_COUNTER__LESS_THAN_OR_EQUAL
Counter__B_COUNTER__EQUAL
Counter__B_COUNTER__GREATER_THAN
Counter__B_COUNTER__GREATER_THAN_OR_EQUAL
Counter__B_COUNTER__SOFTWARE
```

戻り値 なし

副作用 比較出力は、カウンタに書き込まれた値と現在値によってはすぐに変更されることがあります。

void Counter_SetCaptureMode (uint8 captureMode)

説明 キャプチャ モードを設定します。この関数は UDB 実装でキャプチャ モード 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。「制御レジスタ」セクションも参照してください。
```
Counter__B_COUNTER__NONE
Counter__B_COUNTER__RISING_EDGE
Counter__B_COUNTER__FALLING_EDGE
Counter__B_COUNTER__EITHER_EDGE
Counter__B_COUNTER__SOFTWARE_CONTROL
```

戻り値 なし

副作用 なし

void Counter_ClearFIFO (void)

説明 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「関数の説明」セクションの「UDB FIFO」を参照してください。

パラメータ なし

戻り値 なし

副作用 なし

void Counter_Sleep (void)

説明 これは、コンポーネントのスリープを準備するための、好ましいルーチンです。Counter_Sleep() ルーチンは現在のコンポーネント状態を保存します。その後、Counter_Stop() 関数を呼び出し、Counter_SaveConfig() を呼び出して、ハードウェア設定を保存します。

CyPmSleep() または CyPmHibernate() 関数を呼び出す前に Counter_Sleep() 関数を呼び出します。電源管理関数については、『PSoC Creator *システム リファレンス ガイド*』を参照してください。

パラメータ なし

戻り値 なし

副作用 FF 実装では、全低パワー モードで全レジスタの値が保持されます。UDB 実装では、制御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、Counter_Stop() を呼び出さないで Counter_Sleep() Counter_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 Counter_Wakeup (void)

説明 これは、コンポーネントを Counter_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。Counter_Wakeup() 関数は、設定を復元するために Counter_RestoreConfig() 関数を呼び出します。Counter_Sleep() 関数が呼び出される前にコンポーネントが有効になった場合、Counter_Wakeup() 関数がコンポーネントを再度有効にします。

パラメータ なし

戻り値 なし

副作用 最初に Counter_Sleep() または Counter_SaveConfig() 関数を呼び出すことなく Counter_Wakeup() 関数を呼び出すと、予期されない動作につながる可能性があります。

void Counter_Init(void)

説明 カスタマイザの [Configure] (設定) 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。Counter_Start() ルーチンが Counter_Init() 関数を呼び出すので、この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

戻り値 なし

副作用 全レジスタは、カスタマイザの [Configure] (設定) ダイアログの設定に従って、値が設定されます。

void Counter_Enable(void)

説明 ハードウェアの使用を開始し、コンポーネントの動作を開始します。Counter_Start() ルーチンが Counter_Enable() 関数を呼び出すので、この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 モードの両方に対し、カウンタを有効にします。

パラメータ なし

戻り値 なし

副作用 **Enable Mode** パラメータが「Hardware Only」(ハードウェアのみ) に設定されている場合、この関数はカウンタの動作に何ら影響しません。

void Counter_SaveConfig(void)

説明 この関数は、コンポーネントの設定と維持されないレジスタを保存します。この関数は、[Configure] (設定) ダイアログで定義されている、または該当する API で変更される、現在のコンポーネント パラメータ値も保存します。この関数は Counter_Sleep() 関数にて呼び出されます。

パラメータ なし

戻り値 なし

副作用 なし

void Counter_RestoreConfig(void)

説明 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。この関数はまた、コンポーネント パラメータ値を Counter_Sleep() 関数を呼び出した前の状態に復元します。

パラメータ なし

戻り値 なし

副作用 最初に Counter_Sleep() または Counter_SaveConfig() 関数を呼び出すことなくこの関数を呼び出すと、予期されない動作につながる可能性があります。

条件付きコンパイル情報

カウンタ コンポーネント API は、サポートする必要のある複数設定を取り扱うために 2 つの条件付きコンパイル定義を必要とします。API は、FF または UDB ブロックで選択した **Resolution** と **Implementation** パラメータについて条件付きコンパイルする必要があります。定義される 2 つの条件はこれらのパラメータに基づいています。API は決してこれらのパラメータを直接使用してはならず、下に示す 2 つの定義を使用する必要があります。

Counter_DataWidth

DataWidth 定義はビルド時に **Resolution** 値に割り当てられます。これは API 全体で使用され、API 関数の正しいデータ幅でのコンパイルはこの情報にかかっています。

Counter_UsingFixedFunction

UsingFixedFunction 定義は、FF ブロックで提供されるレジスタはコンポーネントが UDB に実装される場合とは異なるため、正しいレジスタ割り当てを行うために主としてヘッダ ファイルで使用されます。FF ブロックは 16 ビットの最大データ幅に制限されるため、場合によっては、この定義は DataWidth 定義と併用されることもあります。

ファームウェア ソースコードの例

PSoC Creator では、[Find Example Project] (サンプル プロジェクトの検索) ダイアログに回路図とサンプル コードを含む数々のサンプル プロジェクトが提供されています。コンポーネント別サンプルを見るには、[Component Catalog] (コンポーネント カタログ) または回路図内のコンポーネントのインスタンスからダイアログを開いてください。一般的なサンプルを見るには、開始ページまたは **[File]** (ファイル) メニューからダイアログを開いてください。必要に応じ、ダイアログ内の **[Filter Options]** (フィルタ オプション) を使って選択できるプロジェクトを絞り込みます。

詳細については、PSoC Creator ヘルプの「Find Example Project」 (サンプル プロジェクトの見つけ方) のトピックを参照してください。

機能説明

一般動作

カウンタ コンポーネントは **Clock Mode** パラメータ設定に従って、一方向 (アップまたはダウン) または両方向にカウントできます。

- 「Up Counter」(アップ カウンタ) または「Down Counter」(ダウン カウンタ) に設定されている場合、コンポーネントは一方向だけにカウントします。カウンタ レジスタは、クロック信号に対するカウント入力の立ち上がりエッジごとに一度インクリメントまたはデクリメントします。
- 「Clock Input + Direction」または「Clock with UpCnt & DwnCnt」に設定されていると、コンポーネントは upCnt、dwnCnt、up_ndown 入力に基づいて両方向にカウントできます。これらの入力については、このデータシートの「入出力接続」セクションに詳しく説明されています。
-

カウンタ オーバーフロー/アンダーフロー

カウンタ オーバーフローとアンダーフローはどのクロック モードでも起き得ます。ステータス レジスタにはオーバーフローまたはアンダーフローが起きたことを示すためのビットがあり、モード レジスタにはこれらの状態で int が生成されるかを制御するためのビットがあります。

Clock Mode	オーバーフロー	アンダーフロー
ダウン カウンタ	未定義。割り込み生成はマスクする必要がある。	カウンタ レジスタが 0 に等しい。
アップ カウンタ	カウンタ レジスタが期間レジスタに等しい。	未定義。割り込み生成はマスクする必要がある。
Clock Input + Direction	カウンタ レジスタが 0xFF、0xFFFF、0xFFFFFF、または 0xFFFFFFFF に等しい。	カウンタ レジスタが 0 に等しい
Clock with UpCnt & DwnCnt	カウンタ レジスタが 0xFF、0xFFFF、0xFFFFFF、または 0xFFFFFFFF に等しい。	カウンタ レジスタが 0 に等しい

カウンタ出力

カウンタ レジスタは監視、リロードできます。tc と comp の 2 つの出力でカウンタ レジスタの現在値を監視し、リロード イベントとして設定できます。詳細については、「入出力接続」セクションを参照してください。

カウンタ レジスタは期間レジスタからリロードします。次の表に、カウント終了とリロードが **Clock Mode** (クロック モード) の各設定に対してどのように機能するかを示します。

Clock Mode	tc 出力がアクティブとなるタイミング	カウンタ リロード値
ダウン カウンタ	カウンタ レジスタが 0 に等しくなった後 1 クロック入力サイクル	カウンタ レジスタが 0 に等しくなったときの期間レジスタの値
アップ カウンタ	カウンタ レジスタが期間レジスタに等しくなった後 1 クロック入力サイクル	カウンタ レジスタが期間レジスタに等しくなったときにカウンタを 0 にリセット
Clock Input + Direction	カウンタ レジスタが 0 にロールオーバーした後 1 クロック入力サイクル	なし – カウンタがロールオーバー
Clock with UpCnt & DwnCnt	カウンタ レジスタが 0 に等しくなった後 1 クロック入力サイクル	なし – カウンタがロールオーバー

comp 出力はカウンタ値と比較値の比較結果を連続的に示します。**Compare Mode** パラメータはすべての標準モードに対して設定できます (「Less Than Or Equal」、「Greater Than」など)。

これは、カウンタがカウントしている間に異なる出力波形を作成するために使用できます。comp 出力は、カウンタのクロック入力に同期されます。

カウンタ入力

キャプチャ動作はハードウェアでもソフトウェアでも実行できます。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、カウンタ コンポーネントを他のコンポーネントに同期できます。カウンタ コンポーネントはイネーブル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。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。

カウンタ割り込み

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 1 つまたは複数の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、割り込みソース、エッジとレベルのどちらをセンスするか、および割り込みソースのクリアの決定をする際に注意深くデザインを検討する必要があります。

カウンタ レジスタ

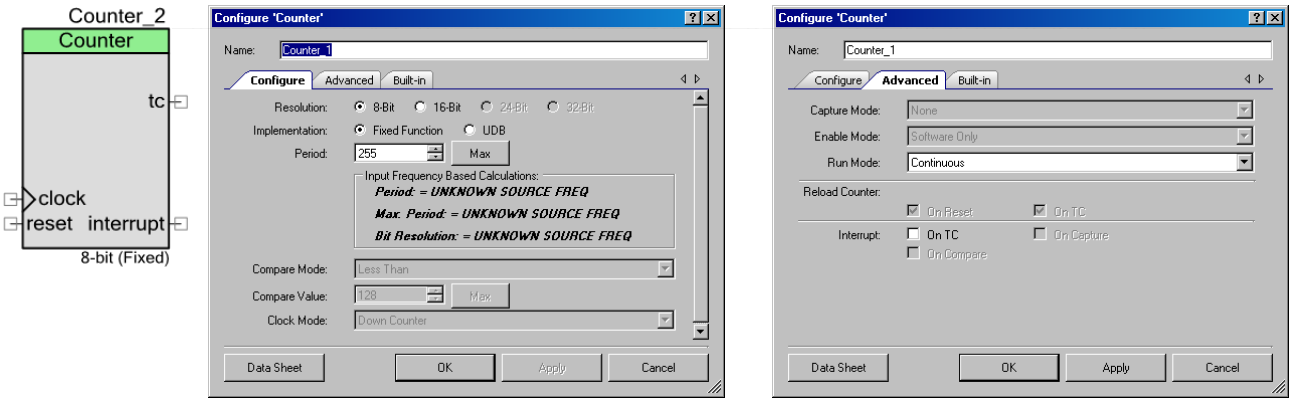
ステータスと制御の 2 種類のレジスタがあります。「レジスタ」セクションを参照してください。

設定

以下のセクションでは、いくつか異なるクロック コンポーネントの設定について説明します。

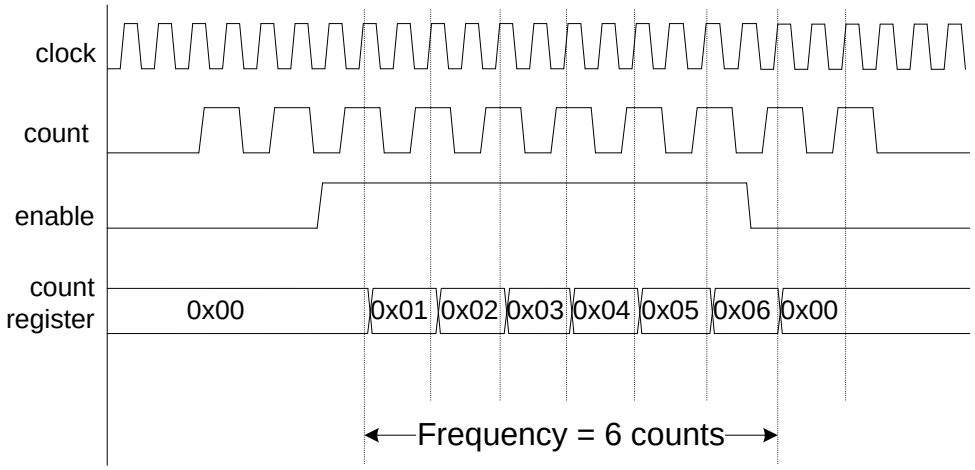
デフォルト設定

カウンタ コンポーネントを PSoC Creator の回路図にドラッグしたときのデフォルト設定は、カウンタ入力の立ち上がりエッジでカウンタ レジスタを減分する 8 ビット、FF カウンタです。下図に、デフォルト コンポーネント記号と [Configure](設定) ダイアログ タブを示します。



下に、デフォルト設定のタイミング図を示します。

図 1. デフォルト設定波形



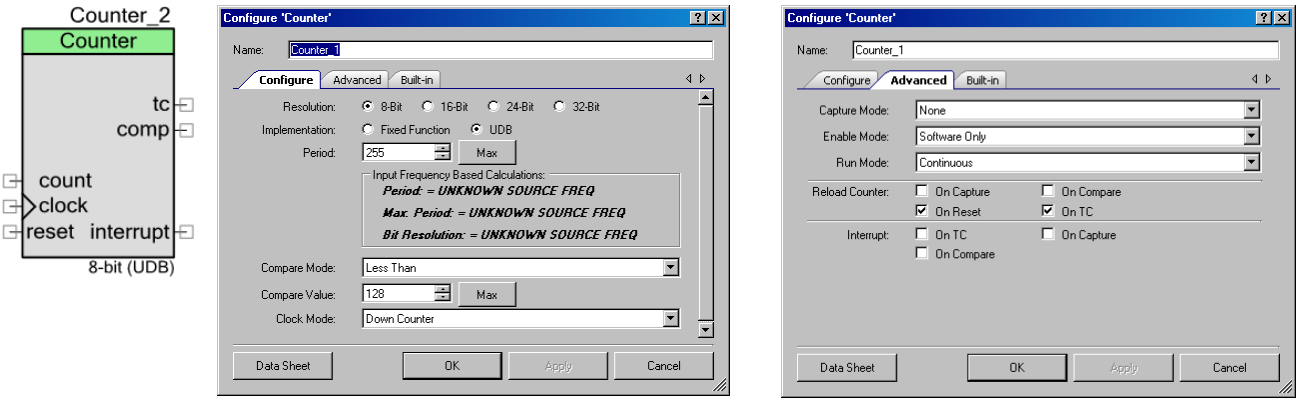
イベント カウンタ設定

Implementation パラメータが「Fixed Function」(固定機能) に設定されているときにカウント入力に印加できる信号には制限があります。このため、コンポーネントを「UDB」に設定すると、イベント カウンタを作成しやすくなります。この構成では、断続的な非同期イベントを検出し、処理してパルスを生成できます。クロック入力はこのカウント入力をサンプルし、**Clock Mode** の設定によってカウンタをインクリメントまたはデクリメントする立ち上がりエッジを形成するために使用します。カウンタ レジスタは CPU でキャプチャし読み取って、発生したイベントの数を得ることができます。

クロック デバイダ設定

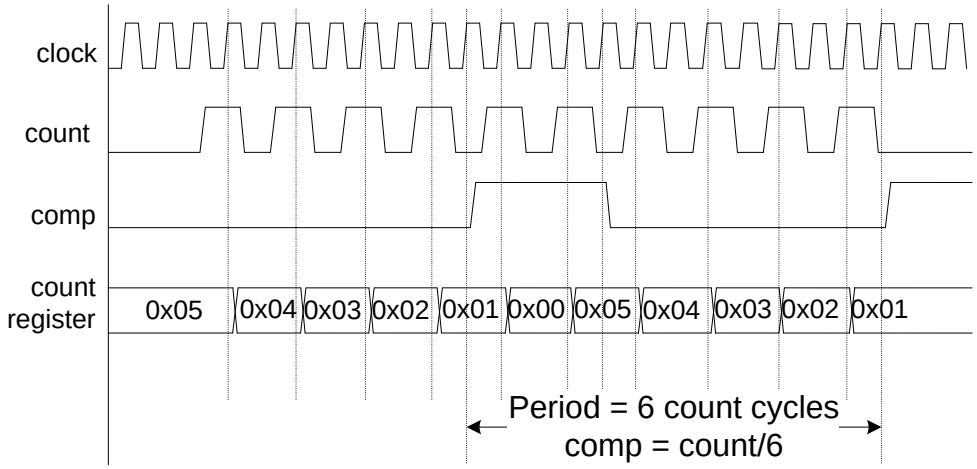
Implementation パラメータの設定を「UDB」に変更することでも、comp 出力を有効にできます。この出力は、プログラム可能な周波数とデューティ サイクルを持つクロック デバイダを作

成するために使用できます。デフォルト設定では、comp 出力は周波数が入力クロックの周波数の 1/256 であるデューティ サイクル クロックの約 50% です。



次の図に、期間が 6、比較値が 2 で **Compare Mode** パラメータが「Less Than」に設定されているサンプル波形を示します。

図 2. クロック デバイダ設定例の波形



周波数カウンタ設定

ハードウェア イネーブルを追加することで、カウンタに周波数カウンタ機能を実装できます。イネーブル入力が既知の期間信号で駆動されている場合、カウント入力上の信号の周波数を得ることができます。**Clock Mode** パラメータを「Down Counter」ではなく「Up Counter」に設定すると演算が簡単になります。

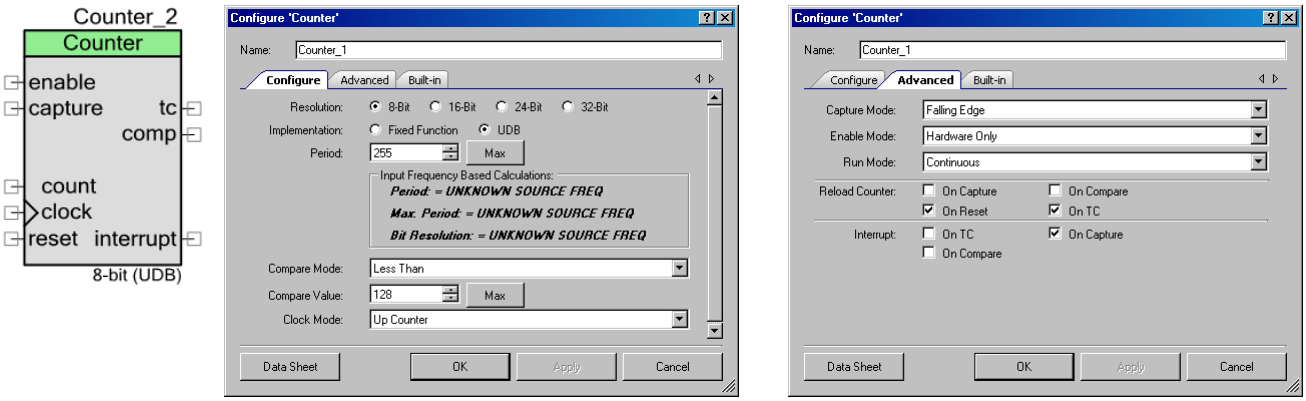
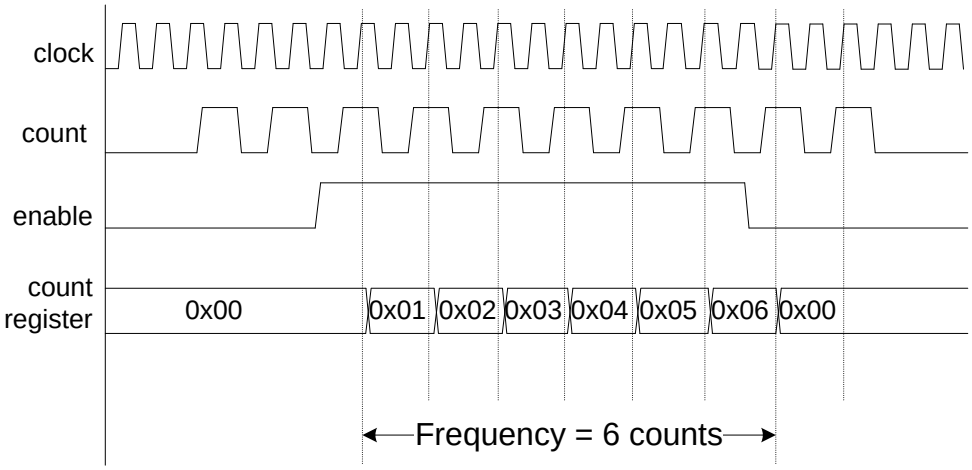


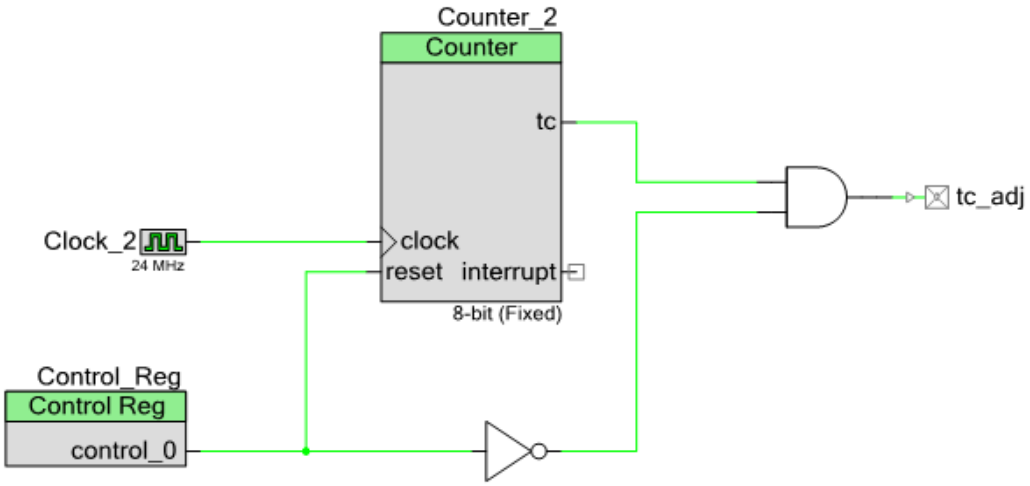
図 3. 周波数カウンタ設定例の波形



固定機能ブロックのリセット

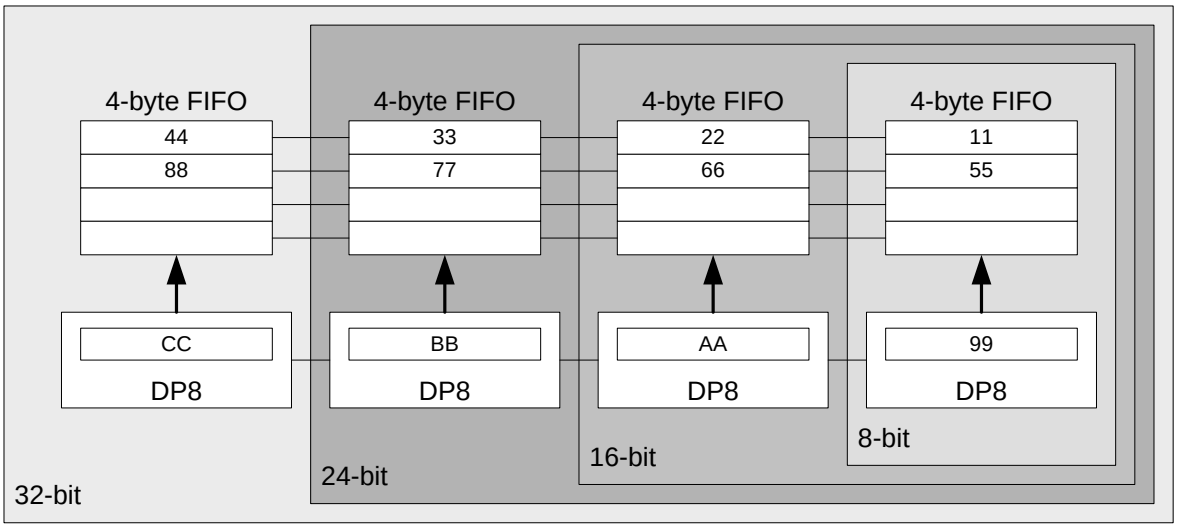
PSoC 3 ES2 シリコン上で、カウンタの FF 実装は UDB 実装とは異なり、リセット時に tc 出力が HIGH になります。UDB 実装では tc は LOW になります。次の回路図に、リセットがアクティブな間に tc LOW を駆動し、同じコンポーネントの UDB 実装と同じ機能を提供する FF 実装を示します。

図 4. PSoC 3 ES2 シリコンのカウント終了調整回路



UDB FIFO

各 UDB datapath には F0 と F1 の 2 つの 8 ビット FIFO があります (詳細は該当するデバイスのデータシートのまたは TRM を参照)。各 FIFO は深さ 4 バイトです。カウンタ UDB 実装は FIFO の 1 つをキャプチャ レジスタとして使用します。他の datapath にあるつ以下の FIFO は 16、24、32 ビット カウンタに使用します。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

すべてのレジスタのアドレス設定するために定義する必要のある定数がいくつかあります。各レジスタ定義には、レジスタ データのポインタか、またはレジスタ アドレスが必要です。異なるコンパイラには異なるエンディアン設定が必要であるため、8 ビットを超えるレジスタ アクセスには各レジスタ用の _PTR 定義を持つ CY_GET_REGX と CY_SET_REGX マクロを使用します。_PTR 定義は生成されたヘッダ ファイルにて提供されます。

ステータス レジスタ

ステータス レジスタは、カウンタ用に定義されたステータス ビットを含む読み取り専用レジスタです。Counter_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 に設定した後、クリアされるまでその状態を保持します。ステータス データはカウンタの入力クロックのエッジでレジ

スタに入れられ、全スティッキー ビットにカウンタのタイミング分解能を与えます。UDB 実装では、ステータス レジスタはバス クロックからクロックされます。FF 実装では、ステータス レジスタはタイマ入力クロックからクロックされます。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。

Counter_Status (UDB 実装)

ビット	7	6	5	4	3	2	1	0
名前	RSVD	FIFO Not Empty	FIFO Full	Capture	Underflow	Overflow	Zero	Cmp
スティッキー	該当なし	偽	偽	真	真	真	真	真

Counter_Status (FF 実装)

ビット	7	6	5	4	3	2	1	0
名前	TC	Capture	Enable	Stop	RSVD	RSVD	RSVD	RSVD
スティッキー	真	真	真	真	該当なし	該当なし	該当なし	該当なし

ビット名	ヘッダ ファイル内の定義	説明
Cmp	Counter_STATUS_CMP	このビットは比較出力が HIGH のときに 1 になります。
Zero	Counter_STATUS_ZERO	このビットはカウンタ値がゼロに等しいときに 1 になります。
Overflow	Counter_STATUS_OVERFLOW	このビットはカウンタ値が期間値に等しいときに 1 になります。
Underflow	Counter_STATUS_UNDERFLOW	このビットはカウンタ値がゼロに等しいときに HIGH になります。
Capture	Counter_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たときに 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	Counter_STATUS_FIFOFULL	このビットは UDB FIFO が 4 エントリとして定義された満杯状態に達したときに 1 になります。
FIFO Not Empty	Counter_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の 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。Counter_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。

カウンタ コンポーネントの割り込み出力は全割り込みソースの OR 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。

Counter_Mode (UDB 実装)

ビット	7	6	5	4	3	2	1	0
名前	RSVD	FIFO Not Empty	FIFO Full	Capture	Underflow	Overflow	Zero	Cmp

Counter_Mode (FF 実装)

ビット	7	6	5	4	3	2	1	0
名前	RSVD	RSVD	RSVD	RSVD	TC	Capture	Enable	Stop

ビット名	ヘッダ ファイル内の定義	割り込み出力オン
Cmp	Counter_STATUS_CMP_INT_MASK	比較
Zero	Counter_STATUS_ZERO_INT_MASK	カウンタ レジスタが 0 に等しい
Overflow	Counter_STATUS_OVERFLOW_INT_MASK	カウンタ レジスタのオーバーフロー
Underflow	Counter_STATUS_UNDERFLOW_INT_MASK	カウンタ レジスタのアンダーフロー
Capture	Counter_STATUS_CAPTURE_INT_MASK	Capture
FIFO Full	Counter_STATUS_FIFOFULL_INT_MASK	UDB FIFO が満杯
FIFO Not Empty	Counter_STATUS_FIFONEMP_INT_MASK	UDB FIFO が空でない

制御レジスタ

制御レジスタを使うと、カウンタの一般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、制御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 制御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。

Counter_Control (UDB 実装)

ビット	7	6	5	4	3	2	1	0
名前	Enable	RSVD	RSVD	Capture Mode [1:0]		Compare Mode[2:0]		

Counter_Mode (FF 実装)

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

ビット名	ヘッダ ファイル内の定義	説明/列挙タイプ
Compare Mode	Counter_CTRL_CMPMODE_MASK	比較モード制御ビットは予期される比較出力動作を定義します。このビット フィールドは、CompareMode パラメータで定義された比較モードでの初期化時に設定されます。 • Counter__B_COUNTER__CM_LESSTHAN • Counter__B_COUNTER__CM_LESSTHANOREQUAL • Counter__B_COUNTER__CM_EQUAL • Counter__B_COUNTER__CM_GREATERTHAN • Counter__B_COUNTER__CM_GREATERTHANOREQUAL
Capture Mode (キャプチャ モード)	Counter_CTRL_CAPMODE_MASK	キャプチャ モード制御ビットは、予期される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、CompareMode パラメータで定義されたキャプチャ モードでの初期化時に設定されます。 • Counter__B_COUNTER__CPTM_NONE • Counter__B_COUNTER__CPTM_RISINGEDGE • Counter__B_COUNTER__CPTM_FALLINGEDGE • Counter__B_COUNTER__CPTM_EITHEREDGE
Enable	Counter_CTRL_ENABLE	ソフトウェア制御下でのカウンティングを有効にします。このビットは、**Enable Mode** パラメータが「Software Only」(ソフトウェアのみ) または「Hardware and Software」(ハードウェアとソフトウェア) に設定されているときにのみ有効です。

カウンタ (分解能 8、16、24、または 32 ビット)

カウンタ レジスタには現在のカウンタ値が入っています。このレジスタは、各種カウント/クロック入力に応えてインクリメントまたはデクリメントされます。このレジスタはいつでも Counter_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 (分解能 8、16、24、または 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タ値が入っています。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。

期間 (分解能 8、16、24、または 32 ビット)

期間レジスタには、Counter_WritePeriod() 関数呼び出しを使って設定し、初期化時に Period パラメータで定義した期間値が入っています。期間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。

比較 (分解能 8、16、24、または 32 ビット)

比較レジスタには、比較 (comp) 出力の状態を決めるために使う比較値が入っています。

DC/AC 電気特性 (FF 実装)

以下の値は、期待されるパフォーマンスを示しており、初期特性データを基にしています。

カウンタの DC 仕様

パラメータ	説明	条件	最小値	典型値	最大値	単位
	ブロックの消費電流	16ビット カウンタ、各入力クロック周波数時	--	--	--	µA
	3 MHz		--	15	--	µA
	12 MHz		--	60	--	µA
	48 MHz		--	260	--	µA
	67 MHz		--	350	--	µA

カウンタの AC 仕様

パラメータ	説明	条件	最小値	典型値	最大値	単位
	動作周波数	PSoC 3	DC	--	67	MHz
	キャプチャ パルス		15	--	--	ns
	分解能		15	--	--	ns
	パルス幅		15	--	--	ns
	パルス幅（外部）		30	--	--	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

DC/AC 電気特性 (UDB 実装)

以下の値は、期待されるパフォーマンスを示しており、初期特性データを基にしています。

「公称ルーティングでの最大」タイミング特性

パラメータ	説明	設定 [1]	最小値	典型値	最大値	単位
f_{clock}	コンポーネントのクロック周波数 [2]	設定 1			45	MHz
		設定 2			40	MHz
		設定 3			35	MHz

[1] 設定:
- 設定 1:
  - 分解能: 8 ビット
  - 実装: 基本アップ/ダウン カウンタ
- 設定 2:
  - 分解能: 8 ビット
  - 実装: カウンタと方向
- 設定 3:
  - 分解能: 16 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)
- 設定 4:
  - 分解能: 24 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)
- 設定 5:
  - 分解能: 32 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)

[2] 時分割多重実装を選択した場合、コンポーネント クロック周波数はデータ レートの 4 倍にする必要があります。

パラメータ	説明	設定 [1]	最小値	典型値	最大値	単位
		設定 4			30	MHz
		設定 5			25	MHz
t_{clockH}	入力クロック HIGH 時間 [3]	該当なし		0.5		$1/f_{clock}$
t_{clockL}	入力クロック LOW 時間[3]	該当なし		0.5		$1/f_{clock}$
入力						
t_{PD_ps}	入力パス遅延、pin to sync [4]	1			STA [5]	ns
t_{PD_ps}	入力パス遅延、pin to sync [6]	2			8.5	ns
t_{PD_si}	同期出力から入力パス遅延 (ルート)	1、2、3、4			STA [5]	ns
t_{I_clk}	clockX と clock の整合	1、2、3、4	0		1	t_{CY_clock}
t_{PD_IE}	入力パス遅延からコンポーネントクロック (エッジ センス入力)	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}	入力パス遅延からコンポーネントクロック (エッジ センス入力)	3、4	$t_{sync} + t_{PD_si}$		$t_{sync} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	入力 HIGH 時間	1、2、3、4	t_{CY_clock} [7]			ns
t_{IL}	入力 LOW 時間	1、2、3、4	t_{CY_clock} [7]			ns

[3] $t_{CY_clock} = 1/f_{clock}$ - 1 クロック期間のサイクル時間。

[4] t_{PD_ps} は、後述される通り静的タイミング解析 (STA) 結果内にあります。ここに挙げられている数値は、多数入力の STA 解析に基づく定格値です。

[5] t_{PD_ps} と t_{PD_si} はルート パス遅延です。ルーティングは動的であるため、これらの値は変化する可能性があり、最大コンポーネントクロックと同期クロックの周波数に直接影響します。値は静的タイミング解析結果になければなりません。

[6] 設定 2 の t_{PD_ps} はデバイスのピンによって定義される固定値です。ここに挙げられている数値はデバイス上で使用可能なすべてのピンの公称値です。

[7] $t_{CY_clock} = 4^* [1/f_{clock}]$ 時分割多重実装を選択した場合。

「すべてのルーティングでの最大」タイミング特性

パラメータ	説明	設定 [1]	最小値	典型値	最大値 [2]	単位
f_{clock}	コンポーネントのクロック周波数 [3]	設定 1			22	MHz
		設定 2			20	MHz
		設定 3			17	MHz
		設定 4			15	MHz
		設定 5			12	MHz
t_{clockH}	入力クロック HIGH 時間 [4]	該当なし		0.5		$1/f_{clock}$
t_{clockL}	入力クロック LOW 時間[4]	該当なし		0.5		$1/f_{clock}$
入力						
t_{PD_ps}	入力パス遅延、pin to sync [5]	1			STA [6]	ns
t_{PD_ps}	入力パス遅延、pin to sync [7]	2			8.5	ns
t_{PD_si}	同期出力から入力パス遅延 (ルート)	1、2、3、4			STA [6]	ns
t_{I_clk}	clockX と clock の整合	1、2、3、4	0		1	t_{CY_clock}

[1] 設定:
- 設定 1:
  - 分解能: 8 ビット
  - 実装: 基本アップ/ダウン カウンタ
- 設定 2:
  - 分解能: 8 ビット
  - 実装: カウンタと方向
- 設定 3:
  - 分解能: 16 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)
- 設定 4:
  - 分解能: 24 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)
- 設定 5:
  - 分解能: 32 ビット
  - 実装: アップ/ダウン カウンタまたは方向付き (Clock with Direction および Clock with UpCnt & DwnCnt など)

[2]「All Routing」(全ルーティング) の最大値は、特性評価ユニット テストを使って取得した経験値に対してテストされた <公称値>/2 を最も近い整数に四捨五入して計算します。この値により、このコンポーネント周波数以下で実行される場合に、ユーザがタイミング条件を気にする必要がなくなります。

[3] 時分割多重実装を選択した場合、コンポーネント クロック周波数はデータ レートの 4 倍にする必要があります。

[4] $t_{CY_clock} = 1/f_{clock}$ - 1 クロック期間のサイクル時間。

[5] t_{PD_ps} は、後述される通り静的タイミング解析 (STA) 結果内にあります。ここに挙げられている数値は、多数入力の STA 解析に基づく定格値です。

[6] t_{PD_ps} と t_{PD_si} はルート パス遅延です。ルーティングは動的であるため、これらの値は変化する可能性があり、最大コンポーネントクロックと同期クロックの周波数に直接影響します。値は静的タイミング解析結果になければなりません。

設定 2 の t_{PD_ps} はデバイスのピンによって定義される固定値です。ここに挙げられている数値はデバイス上で使用可能なすべてのピンの公称値です。

パラメータ	説明	設定 [1]	最小値	典型値	最大値 [2]	単位
t_{PD_IE}	入力パス遅延からコンポーネント クロック (エッジ センス入力)	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}	入力パス遅延からコンポーネント クロック (エッジ センス入力)	3、4	t_{sync} + t_{PD_si}		t_{sync} + t_{PD_si} + $t_{I\ clk}$	ns
t_{IH}	入力 HIGH 時間	1、2、3、4	t_{CY_clock} [8]			ns
t_{IL}	入力 LOW 時間	1、2、3、4	t_{CY_clock} [8]			ns

[8] t_{CY_clock} = 4* [1/f_{clock}] 時分割多重実装を選択した場合。

特性データ用の STA 結果の使用方法

公称ルーティング最大値は、静的タイミング分析 (STA) を使って、複数のテスト パスから収集されます。STA 結果を使って次の方法でデザインごとの最大値を計算できます。

f_{clock} **最大コンポーネント クロック周波数がクロック サマリのタイミング結果に名前付き外部クロックとして表示されます。下図に、*_timing.html* からのクロック リミットの例を示します。**

-Clock Summary

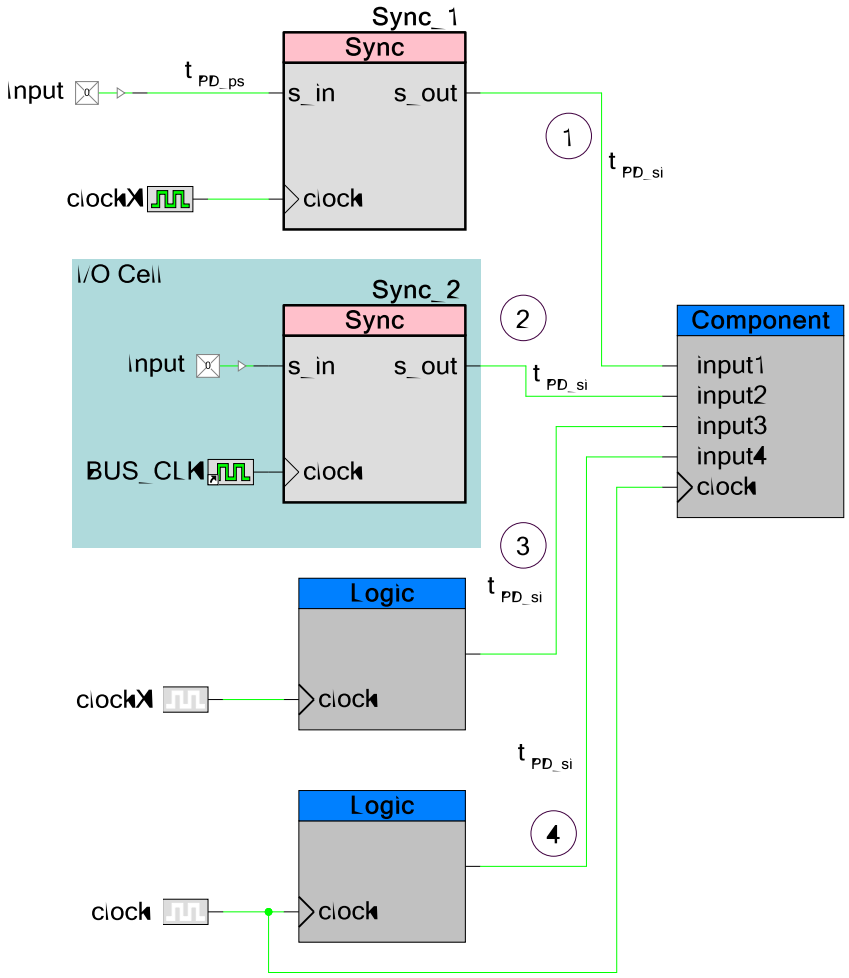
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

入力パス遅延とパルス幅

入力の機能を特性評価するとき、設定の如何にかかわらず、すべての入力は図 5 に示すように 4 つの可能な設定のいずれか 1 つのようになります。

すべての入力は同期される必要があります。同期の仕組みはコンポーネントへの入力のソースによって異なります。システムの機能の仕組みを完全に解釈するには、各入力に対して設定した入力設定とシステムのクロック設定について理解することが必要です。このセクションでは、静的タイミング解析 (STA) 結果を使ってシステムの特性を評価する方法について説明します。

図 5. コンポーネント タイミング仕様に対する入力設定



構成	コンポーネント クロック	同期クロック (周波数)	図
1	master_clock	master_clock	図10
1	clock	master_clock	図8
1	clock	clockX = clock [1]	図 6
1	clock	clockX > clock	図 7
1	clock	clockX < clock	図 9
2	master_clock	master_clock	図 10
2	clock	master_clock	図 8
3	master_clock	master_clock	図 15
3	clock	master_clock	図 13
3	clock	clockX = clock [1]	図 11
3	clock	clockX > clock	図 12
3	clock	clockX < clock	図 14
4	master_clock	master_clock	図 15
4	clock	clock	図 11

[1] クロック周波数は等しいが、立ち上がりエッジの整合は保証されません。

1. 入力はデバイス ピンで駆動され、「sync」コンポーネントで内部的に同期がとられます。このコンポーネントはコンポーネントが使うクロックとは異なる内部クロックを使ってクロッキングされます (内部クロックはすべて master_clock から派生します)。

   この方法で設定された入力を特性評価するとき、clockX はコンポーネント クロックの速度より速い、等しい、遅い可能性があります。これは、図 6、 図 7、図 9、図 10 に示す特性評価パラメータを生成する master_clock に等しい場合もあります。

2. 入力はデバイス ピンで駆動され、master_clock を使ったピンで同期がとられます。

   この方法で設定された入力を特性評価するとき、master_clock はコンポーネント クロックの速度より速いか等しい可能性があります (遅いことはありません)。これは、図 7 と 図 10 に示す特性評価パラメータを生成します。

図 6: 入力設定 1 と 2、同期クロック周波数 = コンポーネント クロック周波数 (clock と clockX のエッジの整合は保証されません)

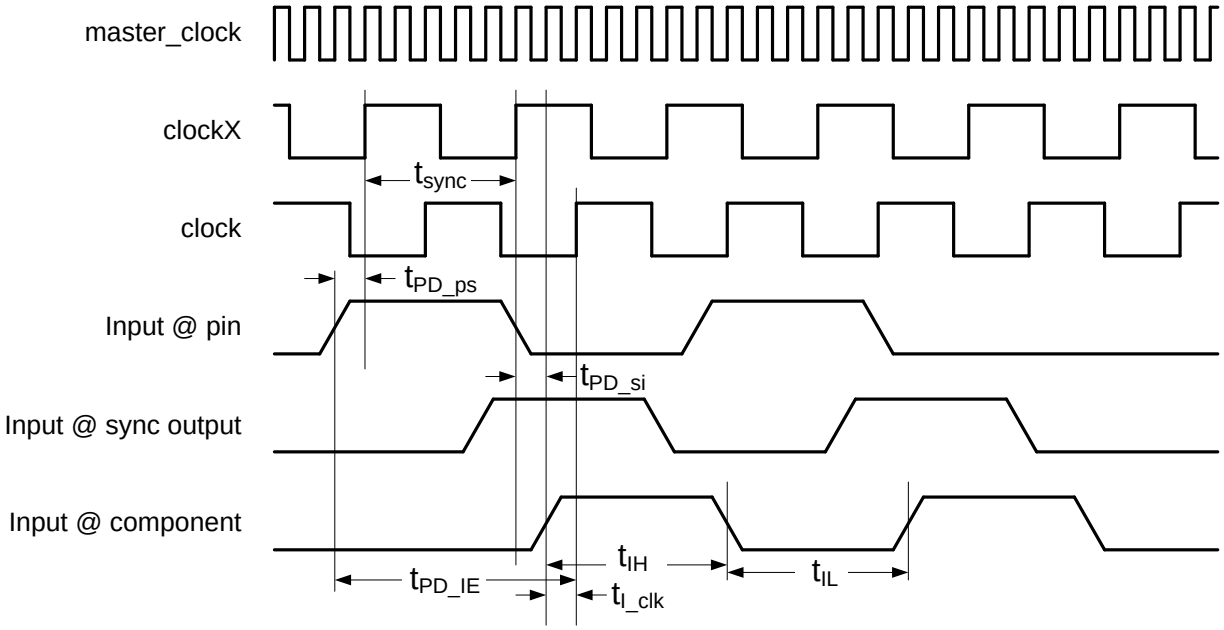


図 7: 入力設定 1 と 2、同期 クロック周波数 > コンポーネント クロック周波数

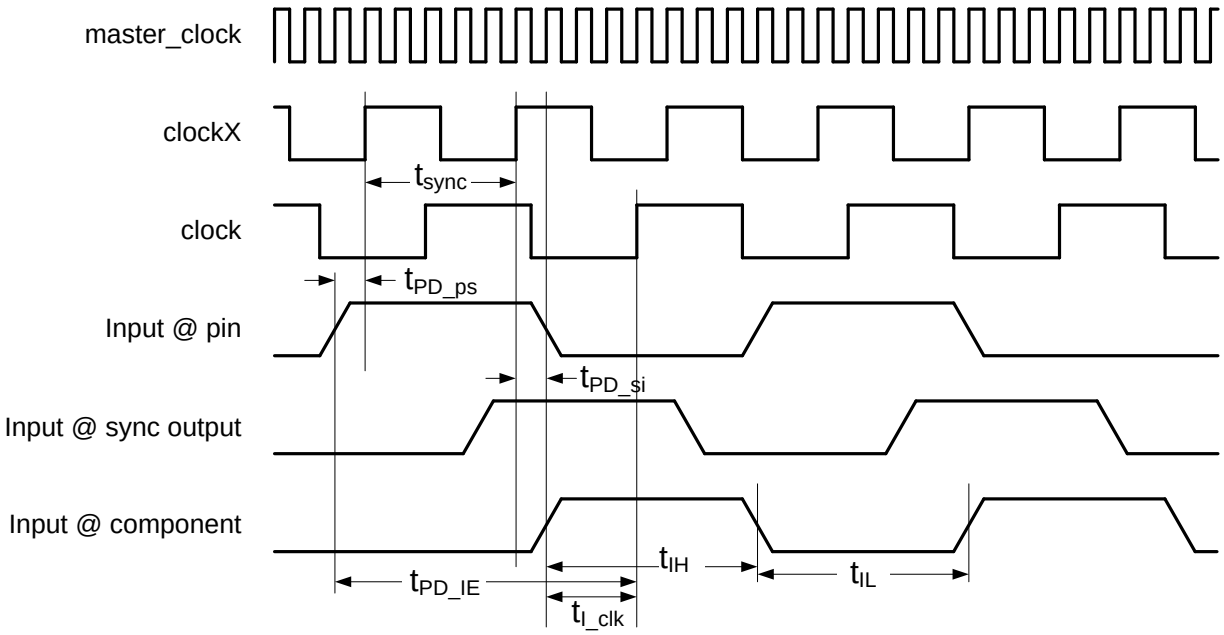


図 8: 入力設定 1 と 2、[同期 クロック周波数 == master_clock] > コンポーネント クロック周波数

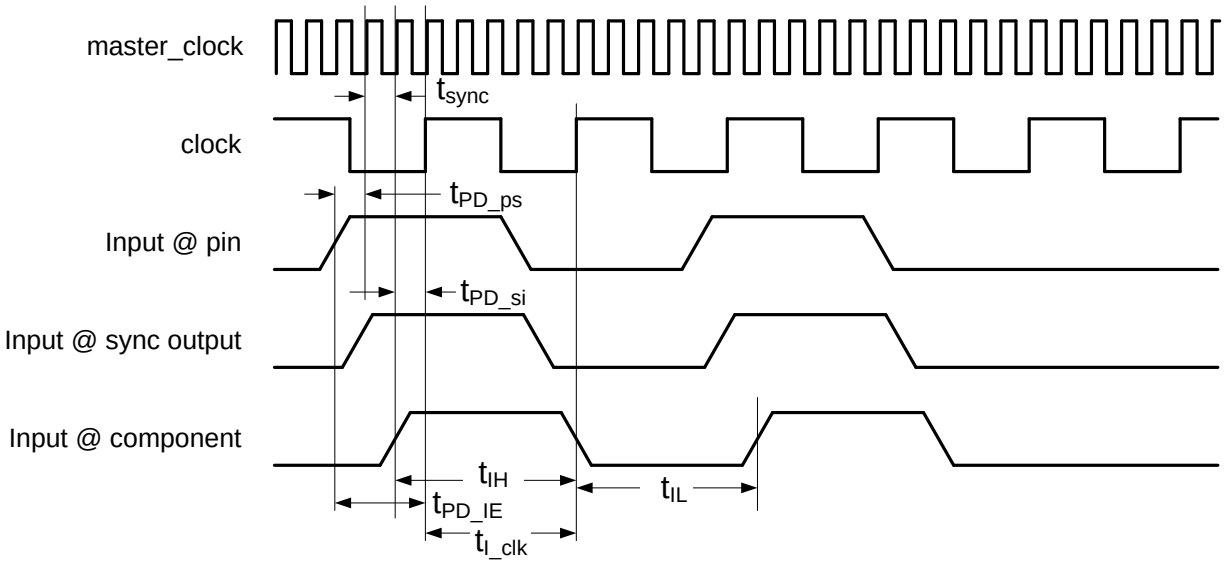


図 9: 入力設定 1、同期 クロック周波数 < コンポーネント クロック周波数

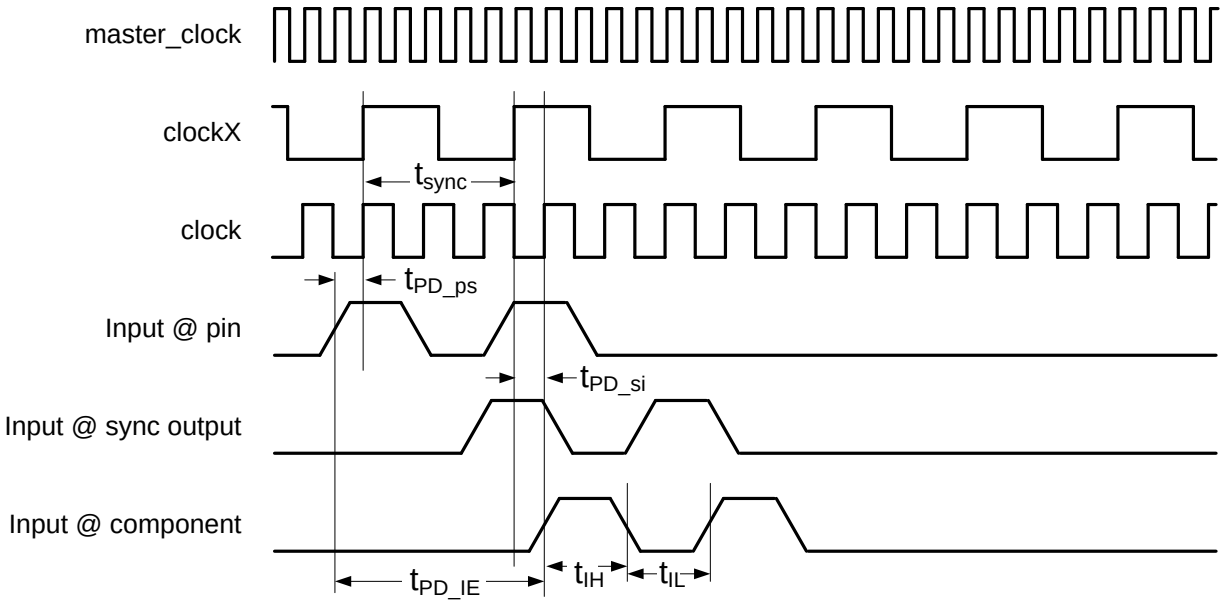
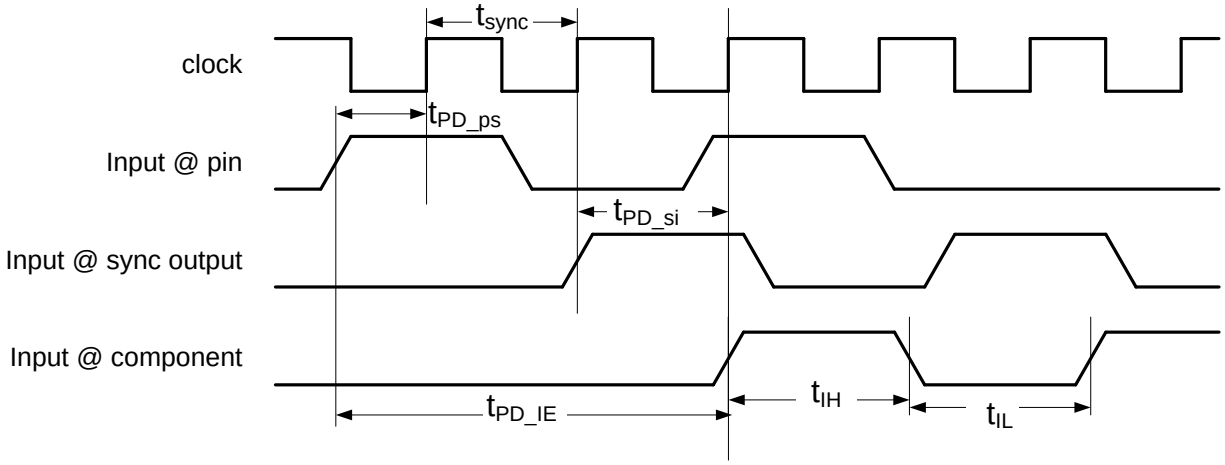


図 10: 入力設定 1 と 2、同期 クロック = コンポーネント クロック = master_clock



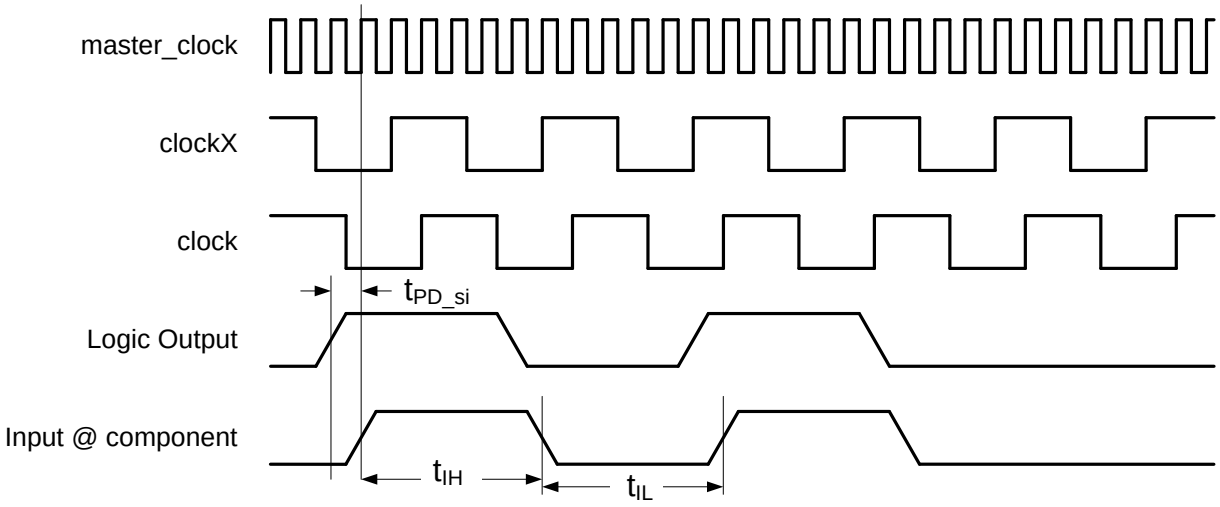
3. 入力は PSoC の内部ロジックで駆動されます。これはコンポーネントが使うクロックとは異なるクロックに基づいて同期されます (内部クロックはすべて master_clock から派生します)。

   この方法で設定された入力を特性評価するとき、同期クロックは図 11、図 12、図 14 に示される特性評価パラメータを生成するコンポーネント クロックの速度より速い、遅い、または等しい可能性があります。

4. 入力は PSoC の内部ロジックで駆動されます。これはコンポーネントが使うクロックのと同じクロックに基づいて同期されます。

この方法で設定された入力を特性評価するとき、同期クロックは図 15 に示される特性評価パラメータを生成するコンポーネント クロックに等しくなります。

図 11: 入力設定 3 のみ、Sync. クロック周波数 = Component Clock Freq. (clock と clockX のエッジの整合は保証されません)



この図は、クロックの静的タイミング解析の理解を助けるものです。デジタル クロック ドメイン内のすべてのクロックは master_clock に同期ですが、同じ周波数を持つ 2 つのクロックの立ち上がりエッジが整合しているとは限りません。このため、静的タイミング解析ツールは、クロックがどのエッジに同期されているか知るすべがなく、最低 1 master_clock サイクルをとることが必要です。これは、t_{PD_si} がシステムの master_clock を制限する効果を持つことを意味します。このパス遅延が長すぎると、Master_clock 設定時間違反が発生します。システムの同期クロックを変更して、master_clock をより遅い周波数で実行する必要があります。

図 12: 入力設定 3、Sync. クロック周波数 > コンポーネント クロック周波数

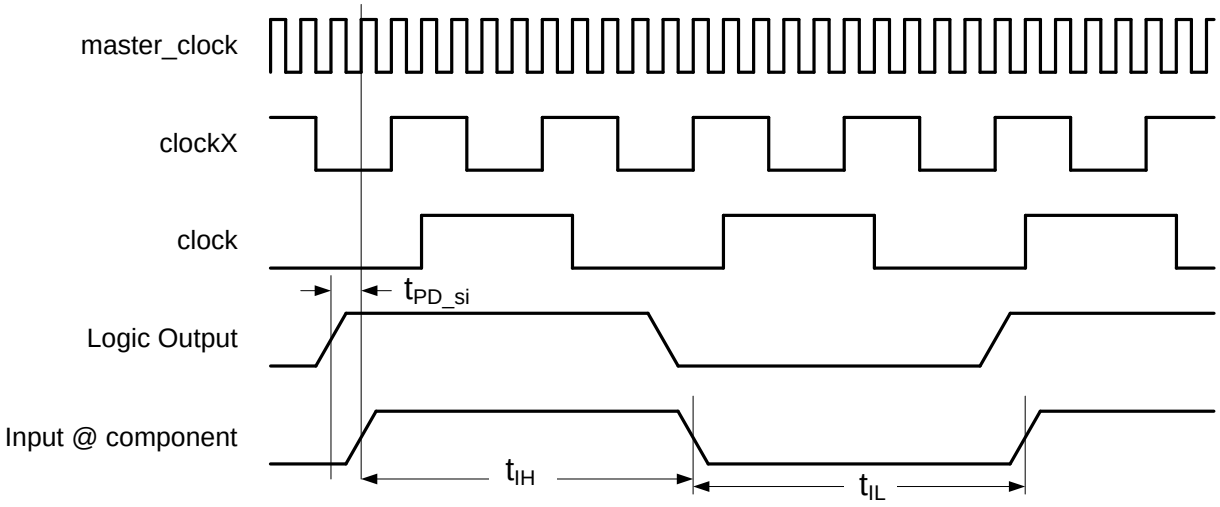


図 11 とほぼ同じ方法で、すべてのクロックは master_clock から派生しています。STA は、この設定での 1 master_clock サイクルに対する master_clock の t_{PD_si} 制限を示します。このパス

遅延が長すぎると、Master_clock 設定時間違反が発生します。システムの同期クロックを変更して、master_clock をより遅い周波数で実行する必要があります。

図 13: 入力設定 3、シンクロナイザ クロック周波数 = master_clock > コンポーネント クロック周波数

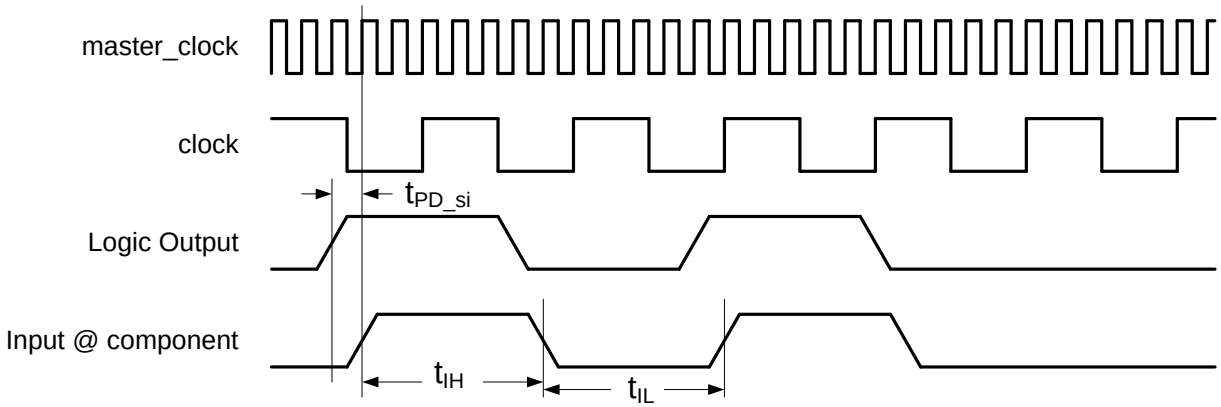


図 14: 入力設定 3、シンクロナイザ クロック周波数 < コンポーネント クロック周波数

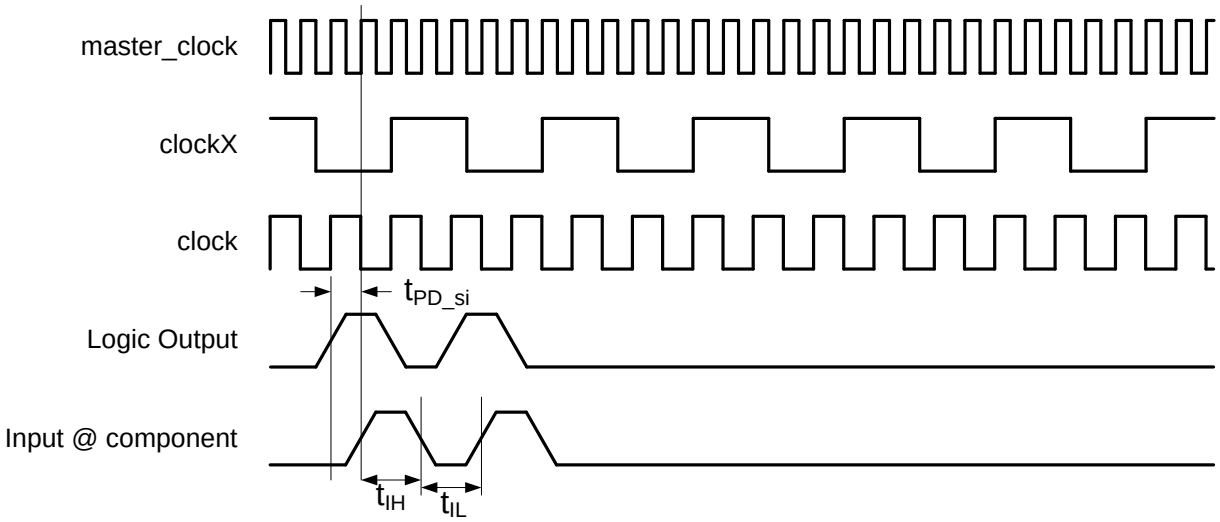
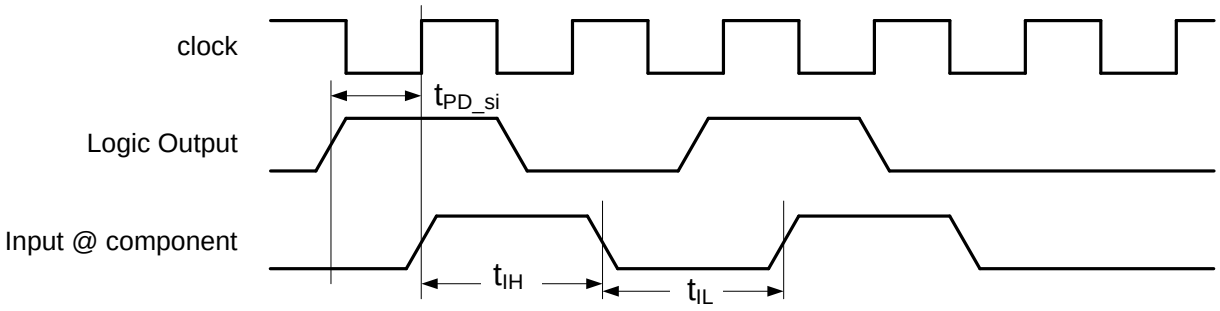


図 11 とほぼ同じ方法で、すべてのクロックは master_clock から派生しています。STA は、この設定での 1 master_clock サイクルに対する master_clock の t_{PD_si} 制限を示します。このパス遅延が長すぎると、Master_clock 設定時間違反が発生します。システムの同期クロックを変更して、master_clock をより遅い周波数で実行する必要があります。

図 15: 入力設定 4 のみ、 シンクロナイザー クロック = コンポーネント クロック



このセクション内でこれまでの図すべてにおいて、実装を理解するために使用する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} (設定 1 と 2 のみ)、t_{PD_si}、 、t_{I_Clk} で定義されます。非常に重要なのは、 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事実です。t_{I_Clk} は STA 結果から派生しませんが、rt_{PD_IE} が登録されるタイミングを示すために使用されます。これは、同期クロックとコンポーネント クロック間のルートの後に残ったマージンです。.

t_{PD_ps} と t_{PD_si} は STA 結果に含まれます。

t_{PD_ps} を見つけるには、*_timing.html* ファイルで定義された入力設定時間を調べてください。この入力のファンアウトは 1 を超える可能性があるので、これらのパス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 時間に定義されます。*_timing.html* ファイルを使用するには net の名前を知っていることが必要です。このパスのファンアウトは 1 を超える可能性があるので、これらのパス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	

出力パス遅延

出力のパス遅延を特性評価するとき、STA 結果のどこでデータを見つけることができるかを知るために、出力の行き先を考慮する必要があります。このコンポーネントでは、すべての出力はコンポーネント クロックに同期されます。出力は次の 2 つのカテゴリのいずれかに分類されます。出力はデバイス内の別のコンポーネントに行くか、ピンからデバイス外部に行きます。前者の場合に、上記のロジック-入力の説明に示されているレジスタ-レジスタ時間を見る必要があります (ソース クロックはコンポーネント クロックです)。後者の場合は、*_timing.html* STA 結果で クロック-出力時間を見ることができます。

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
2.0	全モードでオーバーサンプリングの同期カウンタとして再デザイン。	デバイスのアーキテクチャとツールにより、同期デザインが唯一の実行可能なソリューションであることが証明されました。全モードはまだサポートされているが、オーバーサンプリング用のクロックが必要です。
	固定関数実装から「comp」ターミナルを除去。	実装は比較シュツリョクをサポートしません。ピンは正しく隠されています。
	同期入力	固定関数実装ではすべての入力がブロックの入力時に同期されます。
	Counter_GetInterruptSource() 関数がマクロに変換されました。	Counter_GetInterruptSource() 関数は Counter_ReadStatusRegister() 関数とまったく同じ実装です。コード スペースを節約するために、これは Counter_ReadStatusRegister() 関数のマクロに変換されました。

バージョン	変更の説明	変更の理由 / 影響
	出力がコンポーネント クロックに登録されました。	コンポーネントの出力上の問題を避けるために、すべての出力の同期をとることが必要となります。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。
	補助制御レジスタに書き込む際に重要リージョンを実装しました。	補助制御レジスタに書き込む際に他のプロセス スレッドで変更されないように CyEnterCriticalSection と CyExitCriticalSections 関数を使用します。
	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	

© Cypress Semiconductor Corporation, 2011- 2014. 本文書に記載されている情報は、事前の予告なしに変更される場合があります。サイプレス セミコンダクタ社では、サイプレス製品に統合されている以外の回路の使用については、一切の責任を負いません。また、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、含意することもありません。サイプレス製品は、サイプレスとの明示的な書面による合意がない限り、医療、生命維持、救命、重要な管理、または安全に関わる用途での使用を保証するものではなく、そのような使用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が重大な傷害を負うことが妥当に予測される、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は許可していません。生命維持システム用途にサイプレス製品を使用することは、製造者がそのように使用する上での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ます。

PSoC Designer™、Programmable System-on-Chip™、PSoC Express™ はサイプレス セミコンダクタ社の商標であり、PSoC®はサイプレス セミコンダクタ社の登録商標です。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。

すべてのソースコード（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 セミコンダクタ社（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って保護され、それらの規定に従います。サイプレスが本書面により被免許者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り、該当す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被免許者の製品のみをサポートするカスタム ソフトウェアやカスタム 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、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、本ソースコードのいかなる複製、変更、変換、コンパイル、または表示もサイプレスの明示的な書面による許可がない限り禁止されています。

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を含みますが、それらに限定されません。サイプレスは、本文書に記載した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、本文書に記載した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り生じる一切の責任を負いません。さらに、サイプレスは誤動作や故障によって使用者が重大な傷害を負うことが妥当に予測される、生命維持システムの重要なコンポーネントとしてのサイプレス製品の使用を許可していません。生命維持システム用途にサイプレス製品を使用することは、製造者がそのように使用する上での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、その規定に従います。