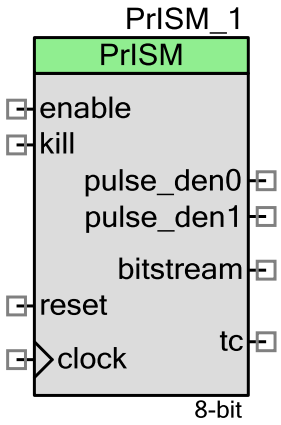
高精度照度信号変調 (PrISM)

2.0

特長



- 2 ～ 32 ビットのプログラマブルなフリッカーなし調光分解能
- 2 つのパルス密度出力
- プログラマブル出力信号密度
- シリアル出力ビットストリーム
- 連続動作モード
- ユーザ設定可能な系列の初期値
- すべての系列長に提供される標準またはカスタム多項式
- パルス密度出力を無効にしてローレベルに強制する Kill 入力
- 他のコンポーネントとの同期動作を提供する Enable 入力
- 他のコンポーネントと同期をとって、系列初期値から再開できる Reset 入力
- 8、16、24、32 ビット系列長のターミナルカウント出力

概要

PrISM (Precision Illumination Signal Modulation, 高精度照度信号変調) コンポーネントは、線形帰還シフトレジスタ (LFSR) を使用して疑似ランダム系列を生成します。系列は、疑似ランダムビットストリームと、2 つまでのユーザ調整可能な疑似ランダムパルス密度を出力します。パルス密度の範囲は 0 ～ 100% です。

LFSR はガロア形式 (モジュラー形式とも呼ばれる) で、提供される最長符号を使用します。PrISM コンポーネントは起動されると、イネーブル信号がハイレベルである限り連続実行されます。PrISM 疑似乱数発生器は、0 以外の任意の初期値(シード)で実行することができます。

PrISM の用途

PrISM コンポーネントには、高輝度 LED 設計に共通の問題である、低周波フリッカーと輻射電磁妨害 (EMI) を大きく低減する変調技術が装備されています。PrISM は、モニタの制御や電源装置など、このメリットを必要とする他のアプリケーションにも便利です。

入出力の接続

ここでは、PrISM のさまざまな入出力接続について説明します。I/O 項目のアスタリスク (*) はその I/O が、説明に挙げられた条件において、回路シンボルに表示されない場合があることを示します。

clock – 入力

クロック入力は、疑似ランダム系列を計算する信号を定義します。

reset – 入力

reset 入力をハイレベルにすると、疑似乱数をシードに戻します。この入力は、起動されたコンポーネンのみに有効であり、他のコンポーネントとの同期動作に利用されます。

kill – 入力

アクティブハイの kill 入力は、PrISM パルス密度出力を無効にし、kill がローレベルになるまで 0 を保持します。

enable – 入力

PrISM コンポーネントネントは起動されると、イネーブル信号がハイレベルで、reset 入力がローレベルである限り動作し続けます。この入力は、他のコンポーネントとの同期動作に利用されます。

pulse_den0/pulse_den1 – 出力

2 つのパルス密度出力が利用できます。2 つとも同じ疑似ランダム系列から生成されます。各出力は、所望のパルス密度と現在の疑似乱数を比較して生成されます。パルス密度タイプを **Less Than or Equal** と設定する場合は、疑似乱数がパルス密度と同じかそれ未満の間は その出力はハイレベルになります。パルス密度タイプを **Greater Than or Equal** と設定する場合は、疑似乱数がパルス密度と同じかそれを超える間は、出力がハイレベルになります。

bitstream – 出力

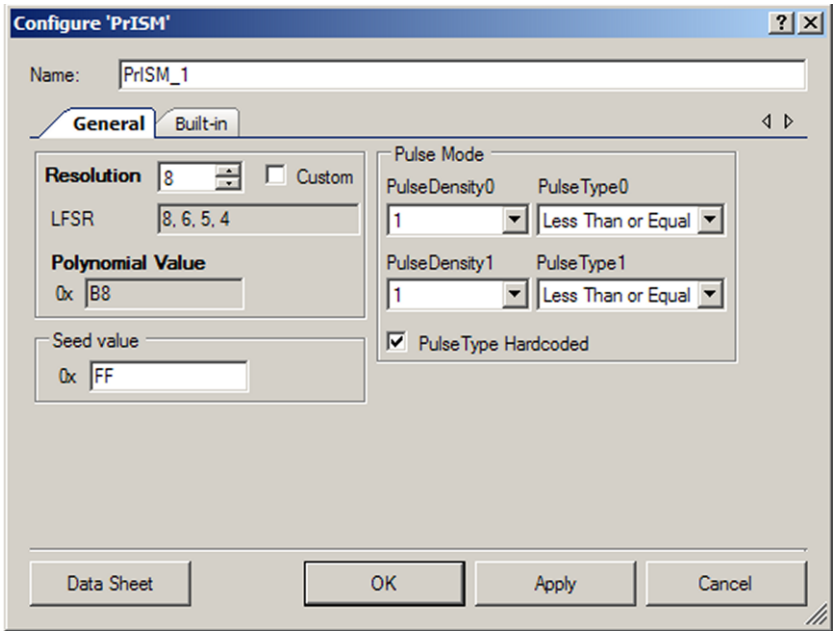
bitstream 出力は、LFSR の LSb を連続して出力します。

tc – 出力*

ターミナル カウント出力は 8、16、24、32 ビット長の PrISM コンポーネントで使用できます。疑似乱数が 0xFF (8 ビット)、0xFFFF (16 ビット)、0xFFFFFF (24 ビット)、0xFFFFFFFF (32 ビット) になる度に、1 クロック期間ハイレベルになります。疑似乱数発生器の各サイクルで 1 回発生します。

コンポーネント パラメータ

PrISM コンポーネントをデザインにドラッグし、ダブルクリックして **Configure** ダイアログを開きます。



PrISM コンポーネントには、次のパラメータがあります。

Resolution

PrISM 最大符号長 (周期) を定義します。最大符号長は、($2^{Resolution} - 1$) です。設定可能な値は 2～32 ビットです。最大符号長は、疑似乱数発生器のシフトレジスタの段数、生成される系列の長さを規定します。長い系列はパルス密度分解能を増やし、輻射 EMI を軽減します。 以下の表に示される最長 周期符号はガロア形式で、PSoC 3 UDB ALU でそれらを使用する前に変換する必要はありません。

Resolution	LFSR		Resolution	LFSR		Resolution	LFSR
2	2, 1		13	13, 12, 10, 9		24	24, 23, 21, 20
3	3, 2		14	14, 13, 11, 9		25	25, 24, 23, 22
4	4, 3		15	15, 14, 13, 11		26	26, 25, 24, 20
5	5, 4, 3, 2		16	16, 14, 13, 11		27	27, 26, 25, 22
6	6, 5, 3, 2		17	17, 16, 15, 14		28	28, 27, 24, 22

Resolution	LFSR		Resolution	LFSR		Resolution	LFSR
7	7, 6, 5, 4		18	18, 17, 16, 13		29	29, 28, 27, 25
8	8, 6, 5, 4		19	19, 18, 17, 14		30	30, 29, 26, 24
9	9, 8, 6, 5		20	20, 19, 16, 14		31	31, 30, 29, 28
10	10, 9, 7, 6		21	21, 20, 19, 16		32	32, 30, 26, 25
11	11, 10, 9, 7		22	22, 19, 18, 17			
12	12, 11, 8, 6		23	23, 22, 20, 18			

LFSR 係数を手動で設定するには:

Resolution を決定します。

Custom チェックボックスを選択します。

LFSR テキストボックスにカンマで区切って係数を入力し、[**Enter**] キーを押します。**Polynomial Value** は自動的に再計算されます。

Polynomial Value は、16 進形式で表示されます。

注 LFSR 係数値は、**Resolution** の値以上にすることはできません。

Polynomial Value

このパラメータは 16 進形式で表示されます。選択した **Resolution** に基づいて、正しい多項式が選択されます。必要に応じてカスタムの多項式を指定することもできます。

Seed value

このパラメータの初期値は、最大値 ($2^{Resolution} – 1$) に設定されています。この値を 0 以外の任意の値に変更することができます。**Seed value** は 16 進形式で表示されます。

注 Resolution を変更すると、**Seed value** は初期値に戻ります。

Pulse Mode

これらのパラメータはコンボ ボックスから選択されます。使用可能な値は、1 ～ $2^{Resolution}$ 単位で ^{Resolution}-1 までです。パルス比較タイプは、**Less Than or Equal** または **Greater Than or Equal** に設定することができます。

PulseType Hardcoded

PulseType Hardcoded パラメータが有効な場合は、リソース (制御レジスタ) を保存しますが、PrISM_SetPulse0Mode() または PrISM_SetPulse1Mode() APIs を使用してパルス タイプを変更できなくなります。

このパラメータが有効の場合は、PrISM_Stop() 関数も使用できなくなります。この場合に PrISM を停止するには、enable 入力を使用します。

ローカルパラメータ (API 用)

これらのパラメータは API で使用され、**Configure** ダイアログ内に表示されます。

- **PolyValue(uint32) –** 16 進形式の多項式の値。初期値は 0xB8h (LFSR= [8,6,5,4]) です。
- **Density0(uint32) –** 16 進形式の density0 の値。
- **Density1(uint32) –** 16 進形式の density1 の値。
- **CompareType0(比較タイプ) –** Density0 用の**パルス タイプ**。**Less Than or Equal** または **Greater Than or Equal**。
- **CompareType1(比較タイプ) –** Density1 用の**パルス タイプ**。**Less Than or Equal** または **Greater Than or Equal**。

クロックの選択

このコンポーネントには、内部クロックがありません。クロックソースを必ず取りつけてください。入力最大周波数は 67MHz です。

配置

PrISM は、UDB アレイ全体に配置され、すべての配置情報は、*cyfitter.h* ファイルを通して API に提供されます。

リソース

リソース	リソースのタイプ				API メモリ(バイト)		ピン(外部入出力ごと)
	データパス セル	PLD	ステータス セル	Control/ Count7 セル	フラッシュ	RAM	
8 ビット	1	3	0	1	423	6	8

8 ビット *	1	3	0	0	423	6	8
16 ビット	2	3	0	1	543	13	8
24 ビット	3	3	0	1	569	23	8
32 ビット	4	3	0	1	569	23	8

* パラメータ **PulseType Hardcoded** が有効の場合

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、各関数へのインターフェースとその説明を示しています。以下のセクションでは、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 "PrISM_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、定数名の接頭辞になります。便宜上、次の表では "PrISM"というインスタンス名を使用します。

関数	機能
PrISM_Start()	Start 関数は、Configure ダイアログで設定された多項式、シード、パルス密度をレジスタに設定します。
PrISM_Stop()	PrISM 計算を停止します。
PrISM_SetPulse0Mode()	Density0 にパルス密度条件を設定します。
PrISM_SetPulse1Mode()	Density1 にパルス密度条件を設定します。
PrISM_ReadSeed()	PrISM Seed レジスタを読み出します。
PrISM_WriteSeed()	シードを PrISM Seed レジスタに書き込みます。
PrISM_ReadPolynomial()	PrISM Polynomial レジスタを読み出します。
PrISM_WritePolynomial()	初期値を PrISM Polynomial レジスタに書き込みます。
PrISM_ReadPulse0()	PrISM Pulse Density0 レジスタを読み出します。
PrISM_WritePulse0()	新しいパルス密度値を PrISM Pulse Density0 値レジスタに書き込みます。
PrISM_ReadPulse1()	PrISM Pulse Density1 レジスタを読み出します。
PrISM_WritePulse1()	新しいパルス密度値を PrISM Pulse Density1 値レジスタに書き込みます。
PrISM_Sleep()	動作を停止し、ユーザ設定を保存します。
PrISM_Wakeup()	ユーザ設定を復元し、有効にします

PrISM_Init()	Configure ダイアログで設定された初期設定に初期化します
PrISM_Enable()	PrISM ブロックの動作を有効にします。
PrISM_SaveConfig()	現在のユーザ設定を保存します。
PrISM_RestoreConfig()	現在のユーザ設定を復元します。

グローバル変数

変数	機能
PrISM_initVar	PrISM の初期化が済んでいるかを示します。変数は、0 に初期化され、最初に PrISM_Start() が呼び出されると 1 にセットされます。これにより、PrISM_Start() ルーチンを最初に呼び出した後で、再初期化を行うことなく、コンポーネントを再起動できます。 コンポーネントの再初期化が必要な場合は、PrISM_Init() 関数を PrISM_Start() または PrISM_Enable() 関数の前に呼び出します。

void PrISM_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。PrISM_Start() は initVar 変数を設定し、PrISM_Init() 関数を呼び出して、PrISM_Enable() 関数を呼び出します。Start 関数は、Configure ダイアログで設定された多項式、シード、パルス密度をレジスタに設定します。PrISM 計算が入力クロックの立ち上がりエッジで開始します。

パラメータ: なし

返り値: なし

注意事項: なし

void PrISM_Stop(void)

機能: PrISM 計算を停止します。出力は一定に保持されます。

パラメータ: なし

返り値: なし

注意事項: **PulseType Hardcoded** パラメータが無効の場合にのみ有効です。

void PrISM_SetPulse0Mode(uint8 pulse0Type)

機能: Density0 にパルス密度条件を設定します。Less Than or Equal(<=) または Greater Than or Equal(>=)。

パラメータ: uint8 pulse0Type: 選択したパルス密度条件

パラメータ値	機能
PrISM_LESSTHAN_OR_EQUAL	疑似乱数が PulseDensity0 レジスタ値と同じかそれ未満の場合、pulse_den0 出力はハイレベルになります
PrISM_GREATERTHAN_OR_EQUAL	疑似乱数が PulseDensity0 レジスタ値と同じかそれを超える場合、pulse_den0 出力はハイレベルになります

返り値: なし

注意事項: **PulseType Hardcoded** パラメータが無効の場合にのみ有効です。

void PrISM_SetPulse1Mode(uint8 pulse1Type)

機能: Density1 にパルス密度条件を設定します。Less Than or Equal(<=) または Greater Than or Equal(>=)。

パラメータ: uint8 pulse1Type: 選択したパルス密度条件

パラメータ値	機能
PrISM_LESSTHAN_OR_EQUAL	疑似乱数が PulseDensity1 レジスタ値と同じかそれ未満の場合、pulse_den1出力はハイレベルになります
PrISM_GREATERTHAN_OR_EQUAL	疑似乱数が PulseDensity1 レジスタ値と同じかそれを超える場合、pulse_den1出力はハイレベルになります

返り値: なし

注意事項: **PulseType Hardcoded** パラメータが無効の場合にのみ有効です。

uint8/16/32 PrISM_ReadSeed(void)

機能: PrISM Seed レジスタを読み出します。

パラメータ: なし

返り値: uint8/16/32: シードレジスタ値

注意事項: なし

void PrISM_WriteSeed(uint8/16/32 seed)

機能: シードを PrISM Seed レジスタに書き込みます。

パラメータ: uint8/16/32) seed: シードレジスタ値

返り値: なし

注意事項: なし

uint8/16/32 PrISM_ReadPolynomial(void)

機能: PrISM Polynomial を読み出します。

パラメータ: なし

返り値: uint8/16/32: 多項式の値

注意事項: なし

void PrISM_WritePolynomial(uint8/16/32 polynomial)

機能: PrISM 多項式を書き込みます。

パラメータ: uint8/16/32 polynomial: 多項式レジスタ値

返り値: なし

注意事項: なし

uint8/16/32 PrISM_ReadPulse0(void)

機能: PrISM PulseDensity0 値レジスタを読み出します。

パラメータ: なし

返り値: uint8/16/32: PulseDensity0 レジスタ値

注意事項: なし

void PrISM_WritePulse0(uint8/16/32 pulseDensity0)

機能: 新しいパルス密度値を PrISM Pulse Density0 値レジスタに書き込みます。

パラメータ: (unit8/16/32) pulseDensity0: パルス密度値。

返り値: なし

注意事項: なし

uint8/16/32 PrISM_ReadPulse1(void)

機能: PrISM Pulse Density1 レジスタを読み出します。

パラメータ: なし

返り値: uint8/16/32: Pulse Density1 レジスタ値

注意事項: なし

void PrISM_WritePulse1(uint8/16/32 pulseDensity1)

機能: 新しいパルス密度値を PrISM Pulse Density1 値レジスタに書き込みます。

パラメータ: uint8/16/32 pulseDensity1: パルス密度値

返り値: なし

注意事項: なし

void PrISM_Sleep(void)

機能: これはコンポーネントをスリープモードにするために好ましいAPIです。PrISM_Sleep() API は、現在のコンポーネントの状態を保存します。次に PrISM_Stop() 関数を呼び出し、PrISM_SaveConfig() を呼び出してハードウェア設定を保存します。

PrISM_Sleep() 関数を CyPmSleep() または CyPmHibernate() 関数を呼び出す前に呼び出します。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: なし

void PrISM_Wakeup(void)

機能: これは、コンポーネントをPrISM_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APIです。PrISM_Wakeup() 関数は PrISM_RestoreConfig() 関数を呼び出して、設定を復元します。PrISM_Sleep() 関数が呼び出される前にコンポーネントが有効であった場合、PrISM_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: あらかじめ PrISM_Sleep() または PrISM_SaveConfig() 関数を呼び出すことなく PrISM_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 PrISM_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。PrISM_Start() API が PrISM_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

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た値が設定されます。

void PrISM_Enable(void)

機能: 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。PrISM_Start() API が PrISM_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。

パラメータ: なし

返り値: なし

注意事項: なし

void PrISM_SaveConfig(void)

機能: この関数は、コンポーネントの設定と保持されないレジスタを保存します。この関数は、Configure 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も保存します。この関数は、PrISM_Sleep() 関数によって呼び出されます。

パラメータ: なし

返り値: なし

注意事項: なし

void PrISM_RestoreConfig(void)

機能: この関数は、コンポーネントの設定と保持されないレジスタを復元します。またコンポーネントのパラメータを 、PrISM_Sleep() 関数を呼び出す前の状態に戻します。

パラメータ: なし

返り値: なし

注意事項: あらかじめ PrISM_Sleep() または PrISM_SaveConfig() 関数を呼び出さずにこの関数を呼び出した場合、予期しない動作をする可能性があります。

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。

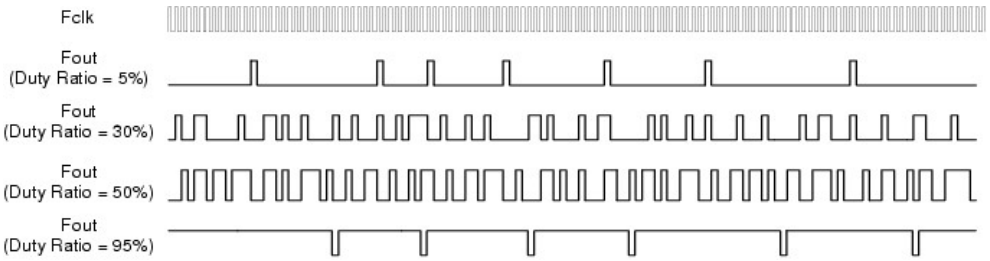
詳しくは、PSoC Creator ヘルプの Find Example Project を参照してください。

機能の詳細

PrISM コンポーネントは、起動され、Enable 入力がハイレベルである限り連続実行されます。PrISM 疑似乱数発生器は、0 以外の任意のシードで開始することができます。これにより、複数の PrISM コンポーネントがある場合、異なる位相から実行することで EMI をさらに低減することができます。reset 入力は、疑似乱数をシードに戻します。アクティブハイの kill 入力は、PrISM パルス密度出力を無効にし、kill がローレベルになるまで 0 を保持します。bitstream 出力は LFSR の LSb を連続的に出力します。

2 つのパルス密度出力が利用できます。2 つとも同じ疑似ランダム系列から派生されます。各出力は、所望のパルス密度と現在の疑似乱数を比較して生成されます。

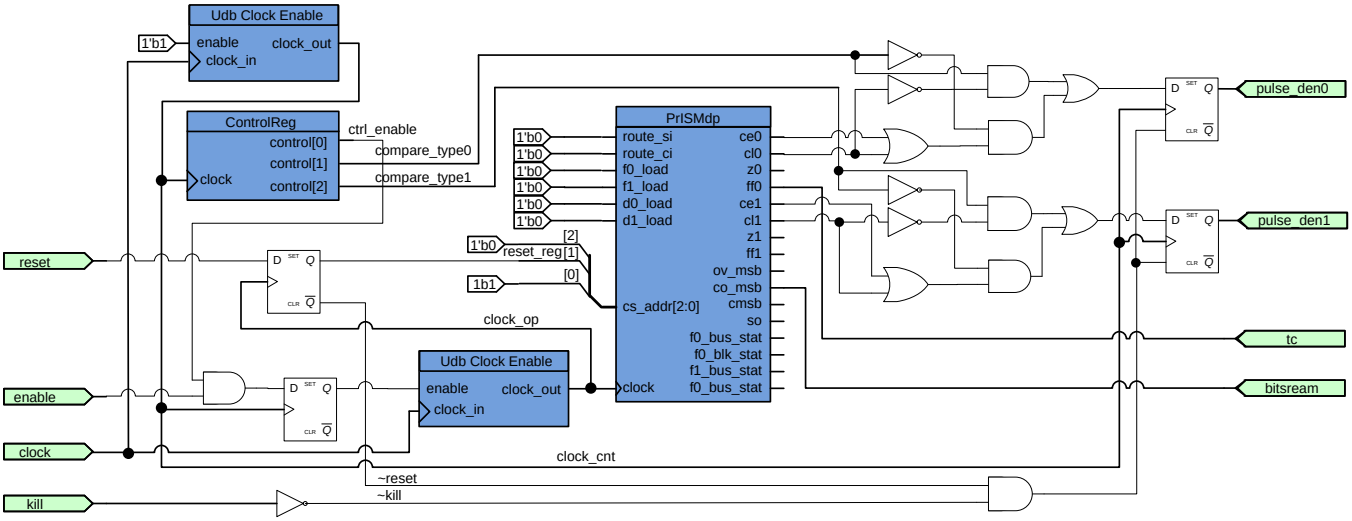
以下のタイミング図は、複数のパルス密度比に基づいた PrISM 出力です。



ブロック図と構成

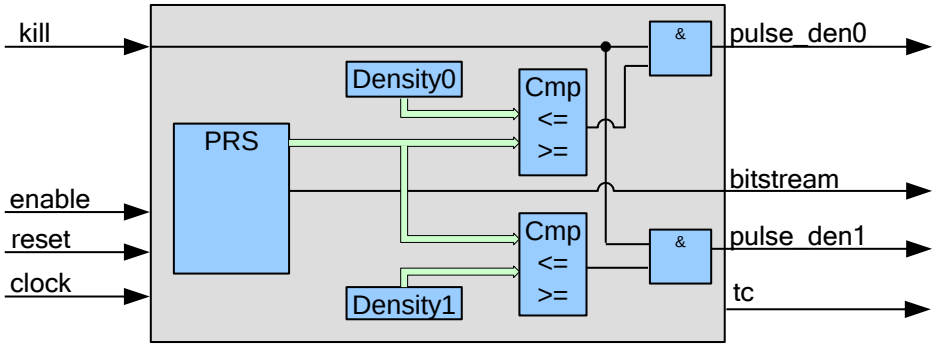
PrISM は UDB 構成としてのみ利用可能です。API は既に説明されており、ここでは PrISM の全体的な実装を定義するためレジスタの説明をします。

以下のブロック図の実装について説明します。



トップレベル アーキテクチャ

2 ～ 32 ビット ハードウェア PrISM コンポーネントは、疑似ランダムカウンタと信号密度値を比較します。カウントが信号密度レジスタの値と同じか未満 (または超過) の場合、コンパレータの出力がアサートされます。



レジスタ

PrISM_CONTROL

ビット	7	6	5	4	3	2	1	0
値	予約済み					compare type1	compare type0	ctrl enable

- ctrl enable: このビットは、前のセクションで説明したすべての内部信号の生成を有効にします。 値は、PrISM_Start() と PrISM_Stop() 関数によって変更することができます。
- compare type0: このビットは、pulse_den0 出力の比較条件を実行します。このビットの値は、コンポーネントの Configure ダイアログのパルス比較条件パラメータで行われた選択で決定されます。さらに値は、PrISM_SetPulse0Mode() 関数によって変更することができます。
- compare type1: このビットは、pulse_den1 出力の比較条件を実行します。このビットの値は、コンポーネントの Configure ダイアログのパルス比較条件パラメータで行われた選択で決定されます。さらに値は、PrISM_SetPulse1Mode() 関数によって変更することができます。

PulseType Hardcoded オプションが選択されている場合は、制御レジスタは使用されません。

PrISM_SEED

ビット	7	6	5	4	3	2	1	0
値	Seed							

- Seed: シード初期値と計算終了時の PRS 剰余を含みます。このレジスタの値は、Configure ダイアログの **Seed value** パラメータによって決定されます。また、PrISM_WriteSeed() 関数で値を変更したり、PrISM_ReadSeed() で読み込むことができます。

PrISM_SEED_COPY

ビット	7	6	5	4	3	2	1	0
値	Seed_Copy							

- Seed_Copy: reset 入力がアクティブになる場合に PrISM_SEED レジスタに自動的に書き込まれる シードを含みます。このレジスタの値は、Configure ダイアログの **Seed value** パラメータによって決定され、PrISM_WriteSeed() 関数が呼び出されると自動的に更新します。

PrISM_POLYNOM

ビット	7	6	5	4	3	2	1	0
値	Polynomial							

- Polynomial: 選択した分解能に基づいて選択された正しい多項式。PrISM_WritePolynomial() 関数で値を変更し、PrISM_ReadPolynomial() 関数で読み出します。

PrISM_DENSITY0

ビット	7	6	5	4	3	2	1	0
値	Pulse density0							

- Pulse density0 は、PrISM pulse_den0 出力の値を決定します。このレジスタの値は、Configure ダイアログの **PulseDensity0** パラメータで決定されます。値を変更するには、PrISM_SetPulse0Mode() 関数を使用します。

PrISM_DENSITY1

ビット	7	6	5	4	3	2	1	0
値	Pulse density1							

- Pulse density1 は、PrISM pulse_den1 出力の値を決定します。このレジスタの値は、Configure ダイアログの **PulseDensity1** パラメータで決定されます。値を変更するには、PrISM_SetPulse1Mode() 関数を使用します。

リファレンス

PRS コンポーネントのデータシートも参照してください。

DC/ AC 電気的特性

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			66	MHz
		16ビット			46	MHz
		24ビット			42	MHz
		32ビット			38	MHz
t_{clockH}	入力クロックハイレベル時間 [1]	該当せず		0.5		$1/f_{CLOCK}$
t_{clockL}	入力クロックローレベル時間 [1]	該当せず		0.5		$1/f_{CLOCK}$
Inputs (入力)						
t_{PD_ps}	入力配線遅延、pin to sync [2]	1			STA [3]	ns
t_{PD_ps}	入力配線遅延、pin to sync [4]	2			8.5	ns
t_{PD_si}	Sync出力から入力への遅延(配線)	1,2,3,4			STA [3]	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

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

[2] t_{PD_ps} は、後述される静的タイミング解析 (STA) の結果にあります。ここに挙げた数字は、STA 解析に基づく公称値です。

[3] t_{PD_ps} と 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。

[4] 構成 2 における[4] 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。

"全配線の最大値"タイミング特性

記号	項目	構成	Min	Typ	Max[1]	単位
f_{CLOCK}	コンポーネント クロック周波数	8ビット			40	MHz
		16ビット			23	MHz
		24ビット			21	MHz
		32ビット			19	MHz
t_{clockH}	入力クロックハイレベル時間 [2]	該当せず		0.5		$1/f_{CLOCK}$
t_{clockL}	入力クロックローレベル時間 [2]	該当せず		0.5		$1/f_{CLOCK}$
入力						
t_{PD_ps}	入力配線遅延、pin to sync [3]	1			STA [4]	ns
t_{PD_ps}	入力配線遅延、pin to sync [5]	2			8.5	ns
t_{PD_si}	Sync出力から入力への遅延(配線)	1,2,3,4			STA [4]	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

[1]全配線の最大値は、公称値を 2 で割った値を最も近い整数に切り上げ/切り下げたものです。この値によって、コンポーネントがこの周波数以下で動作している場合、タイミング条件を満たすか検討する必要がなくなります。

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

[3] t_{PD_ps} は、後述される静的タイミング解析 (STA) の結果にあります。ここに挙げた数字は、STA 解析に基づく公称値です。

[4] t_{PD_ps} と 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。

[5] 構成 2 における[5] 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

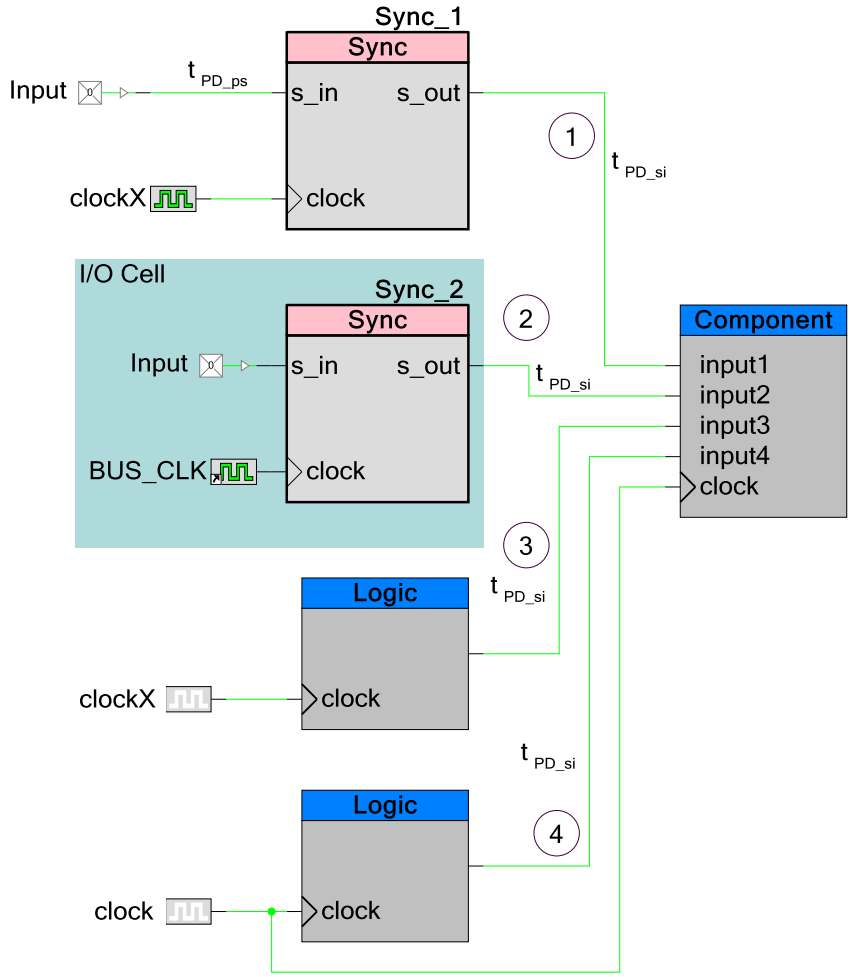
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

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 1 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 1. 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6
1	clock	master_clock	図4
1	clock	clockX = clock [1]	図2
1	clock	clockX > clock	図3
1	clock	clockX < clock	図5
2	master_clock	master_clock	図6
2	clock	master_clock	図4

[1] 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11
3	clock	master_clock	図9
3	clock	clockX = clock [1]	図7
3	clock	clockX > clock	図8
3	clock	clockX < clock	図10
4	master_clock	master_clock	図11
4	clock	clock	図7

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。これは、図 2、図 3、図 5、図 6 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)。これは、図 3 と図 6 に示す特性解析パラメータが生成されます。

図 2. 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)

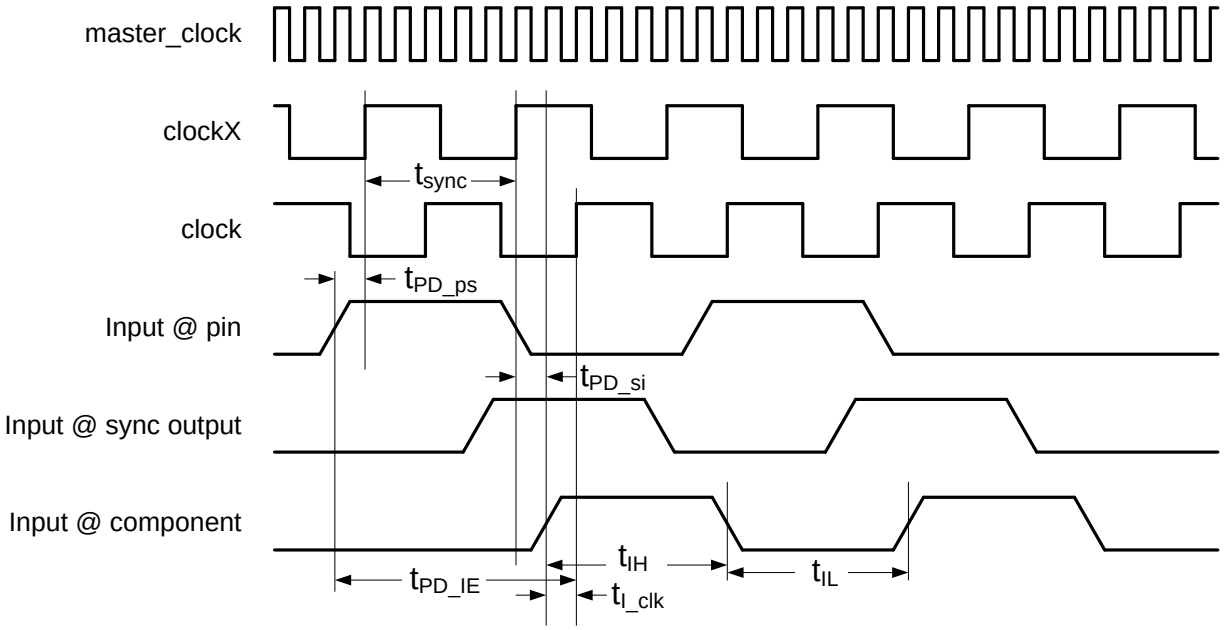


図 3. 入力構成 1 および 2、Sync Clock Frequency > Component Clock Frequency

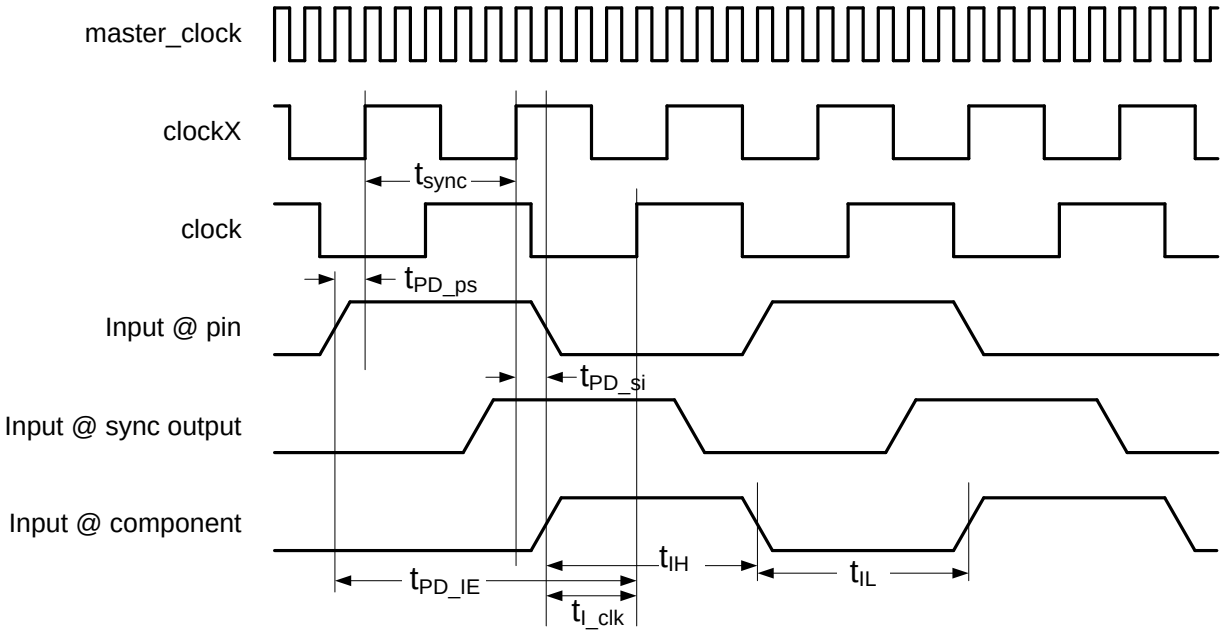


図 4. 入力構成 1 および 2、[Sync Clock Frequency == master_clock] > Component Clock Frequency

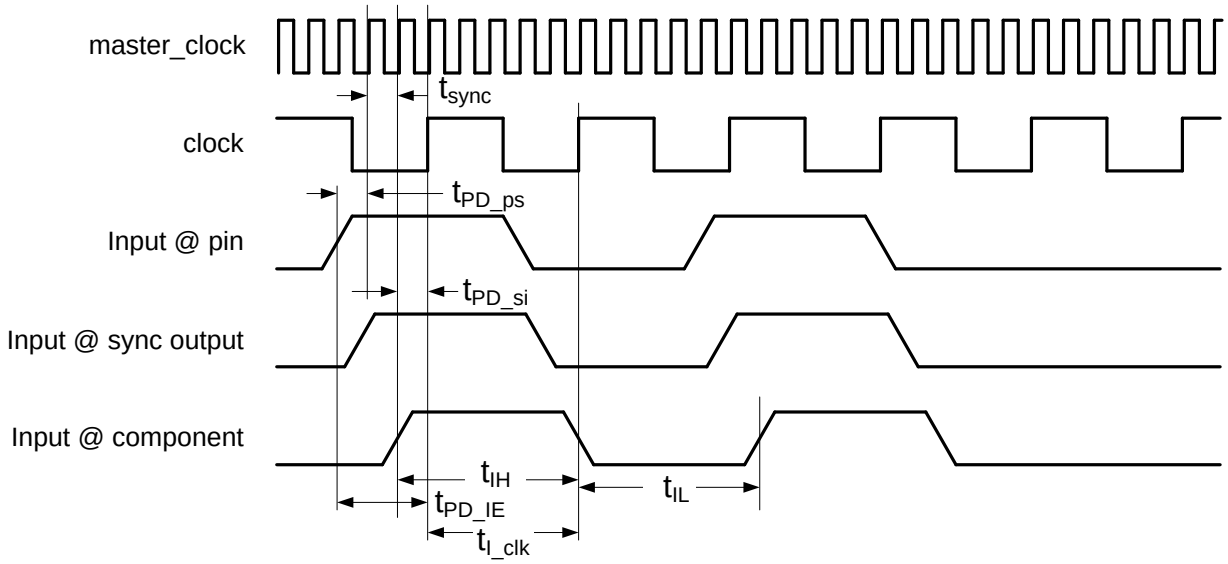


図 5. 入力構成 1、Sync Clock Frequency < Component Clock Frequency

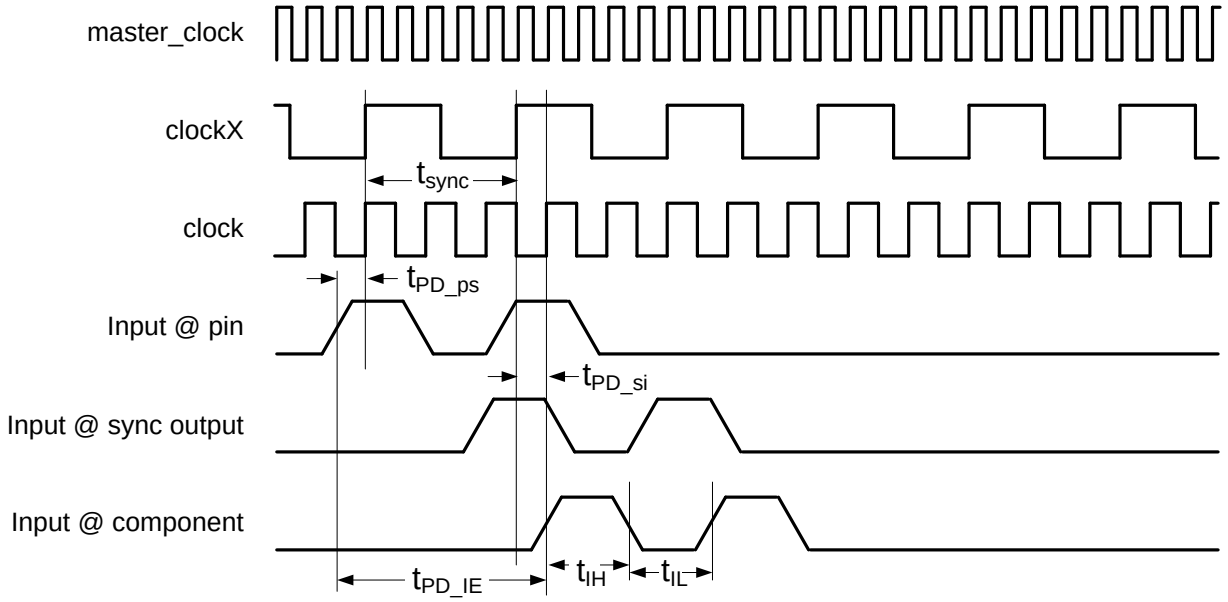
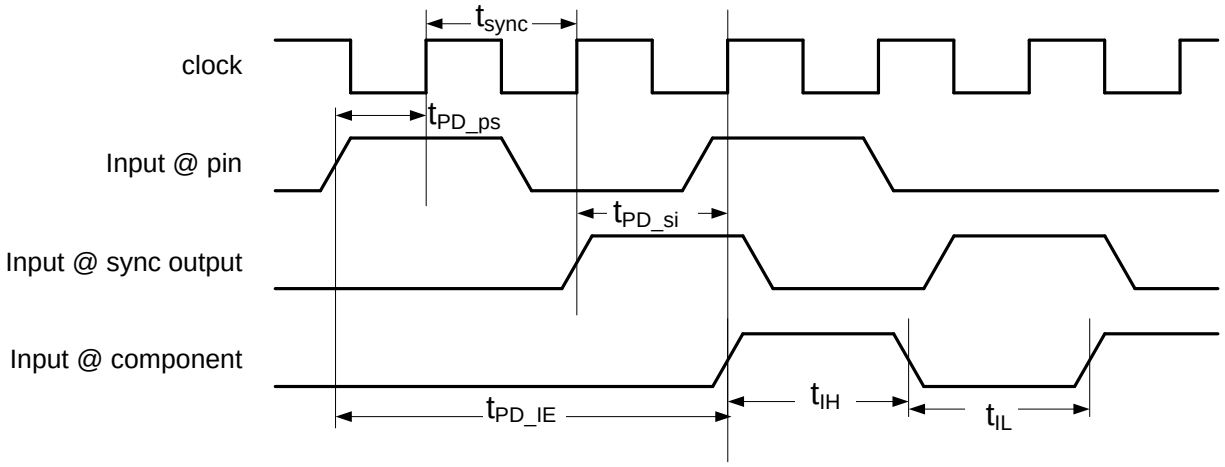


図 6. 入力構成 1 および 2、Sync Clock = Component Clock = master_clock



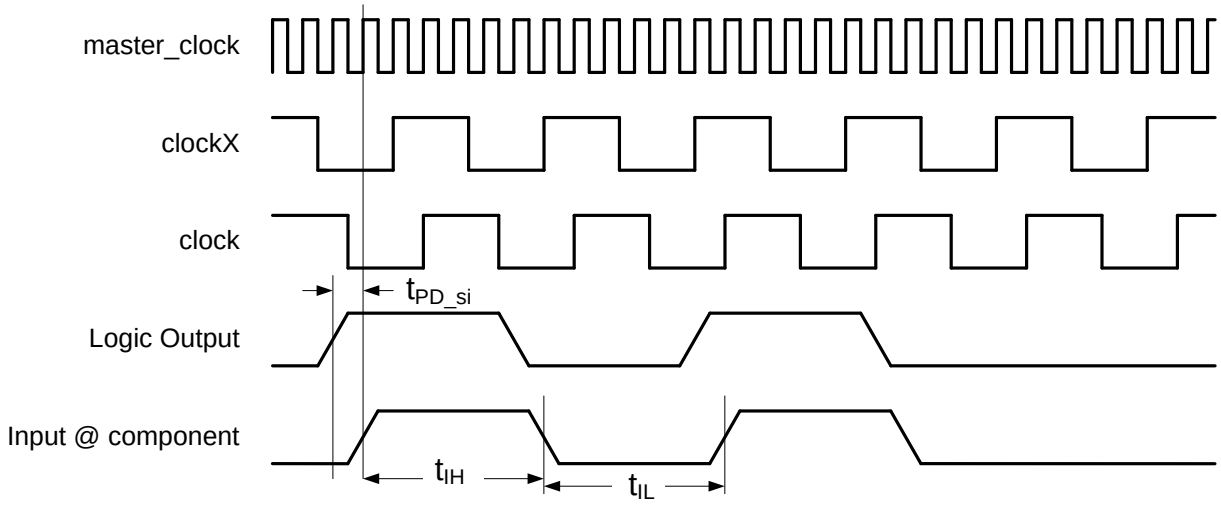
3. 入力は PSoC 内部の Logic に駆動されます。これは Component が使用するクロックとは異なるクロックをベースにして同期しています(すべての内部クロックは master_clock から派生)。

   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。これは、図 7、図 8、図 10 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。これは、図 11 に示す特性解析パラメータが生成されます。

図 7. 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 8. 入力構成 3、Sync Clock Frequency > Component Clock Frequency

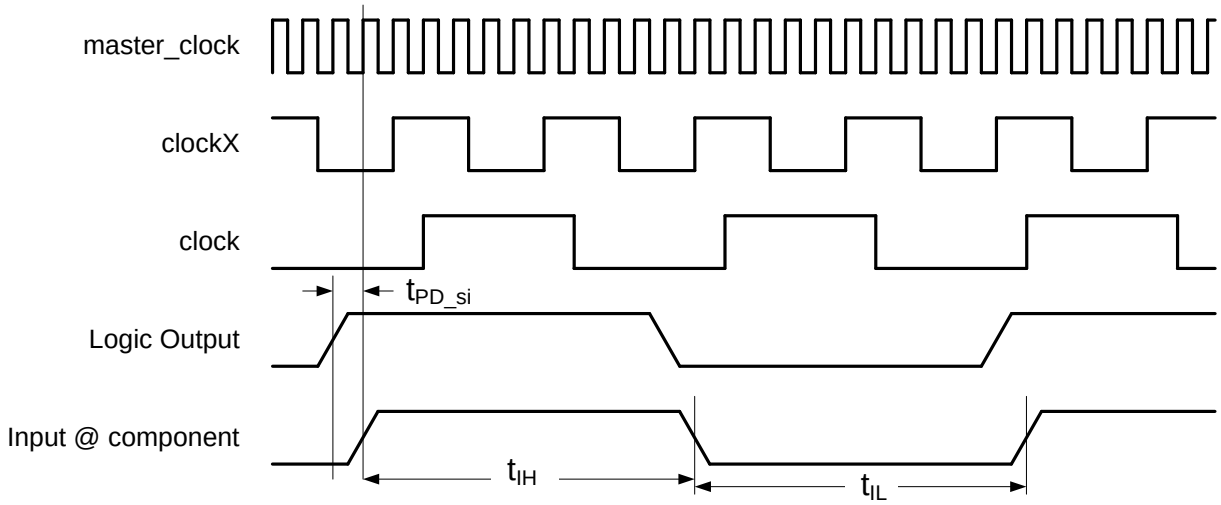


図 7 と同様の方法で、すべてのクロックは master_clock から派生します。STA は 1 master_clock サイクルに関する t_{PD_si} の制限を示します。この配線の遅延が大きいと、master_clock セットアップ時間に違反します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 9. 入力構成 3、Synchronizer Clock Frequency = master_clock > Component Clock Frequency

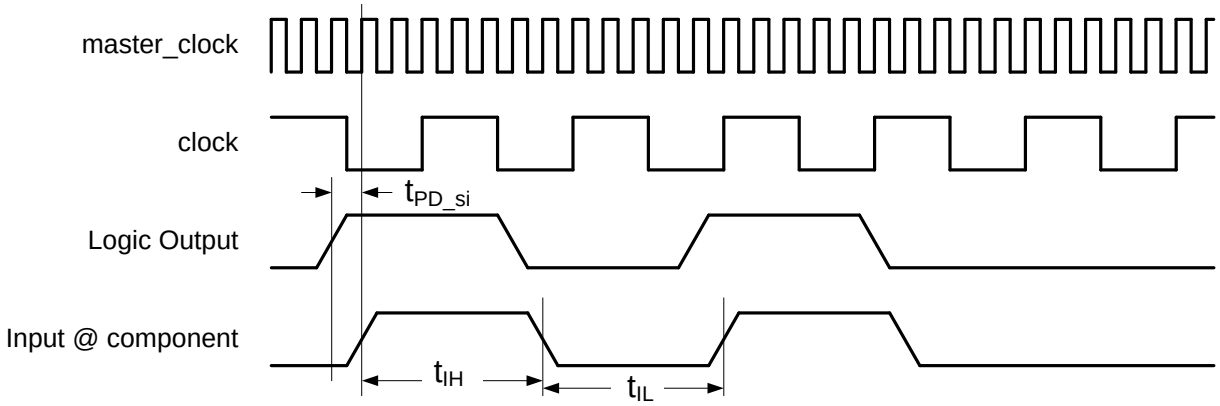


図 10. 入力構成 3、Synchronizer Clock Frequency < Component Clock Frequency

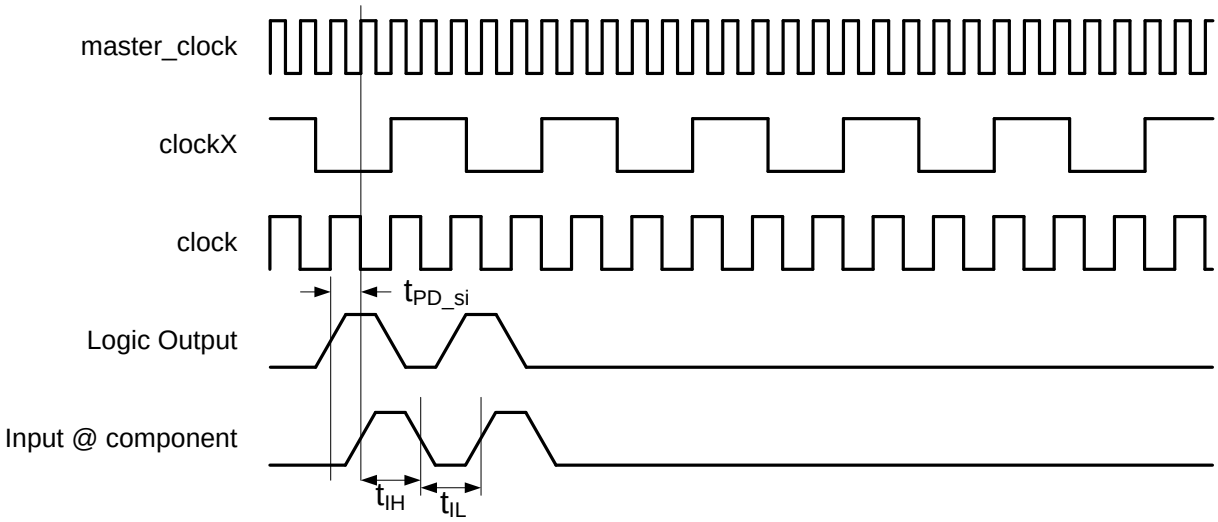
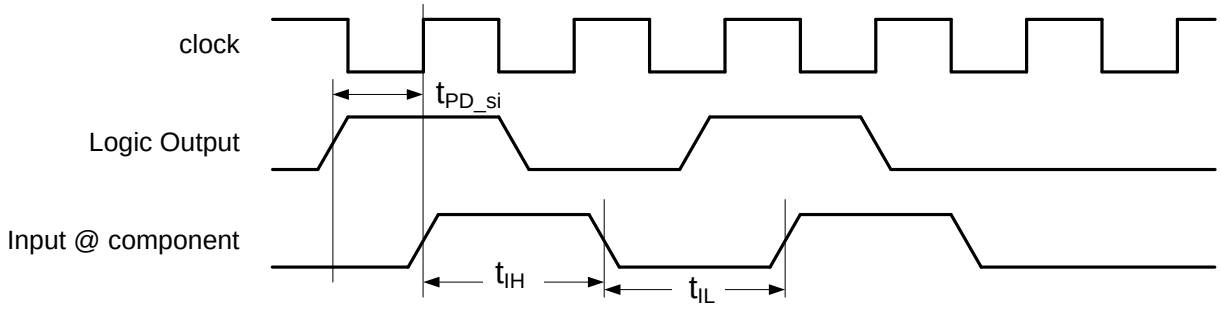


図 7 と同様の方法で、すべてのクロックは master_clock から派生します。STA は 1 master_clock サイクルに関する t_{PD_si} の制限を示します。この配線の遅延が大きいと、master_clock セットアップ時間に違反します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 11. 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} および t_{SYNC} (設定 1 と 2 の場合のみ)、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} と 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* 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。

コンポーネントの変更

ここでは、過去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.a	データシートのマイナーな編集と更新	
2.0	パルス遅延出力はグリッチをなくすために登録されます。	任意の組合せの出力は、信号間の配置や遅延によって誤動作する場合があります。グリッチを除去するには、登録する必要があります。
	enable および reset 入力は最大速度の動作を向上させるために登録されます。	これらの入力には組合せの使用が含まれているので Creator によって自動的に登録 されませんでした。エラーがあります。登録は最大速度を向上させ、グリッチを防ぎます。
	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	

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。