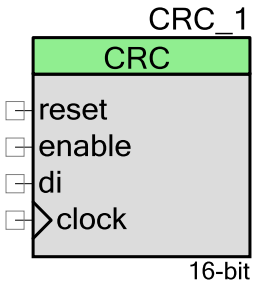
巡回冗長チェック(CRC)

2.10

特長



- 1～64 ビット
- 時分割多重化モード
- シリアル ビット ストリーム入力用のクロックおよびデータが必要
- シリアルデータ入力、パラレル出力
- 標準 [CRC-1 (パリティ ビット)、CRC-4 (ITU-T G.704)、CRC-5-USB など] 、またはカスタム多項式
- 標準またはカスタムのシード値
- イネーブル入力が、他のコンポーネントとの同期動作を提供

概要説明

巡回冗長チェック (CRC) コンポーネントのデフォルトの使用方法は、あらゆる長さのシリアル ビット ストリームから CRC を計算することです。データ クロックの立ち上がりエッジで、入力データがサンプリングされます。CRC 値は、開始前に 0 に設定されるか、オプションで初期値をシードとして使用できます。ビットストリームが完了すると、計算された CRC 値が読み出されます。

CRC を使用するとき

デフォルトの CRC コンポーネントは、送信やストレージ中のデータの変化を検出するため、チェックサムとして使用します。CRC は、バイナリ ハードウェアに簡単に実装でき、数学的に簡単に解析可能であり、伝送路のノイズによって引き起こされる一般的なエラーの検出に特に適しているといった点から、よく利用されています。

入出力接続

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

clock – 入力

CRC は、CRC の計算に使用される、シリアル ビットストリームを提供するデータ入力を必要とします。シリアル データ入力を正しくサンプリングするために、データ クロック入力も必要です。データ クロックの立ち上がりエッジで、入力データがサンプリングされます。

reset – 入力

リセット入力は、CRC を非同期リセットするための信号を定義します。

enable – 入力

CRC コンポーネントは、スタートされた後、イネーブル信号が HIGH である限り動作し続けます。この入力が、他のコンポーネントとの同期動作を提供します。

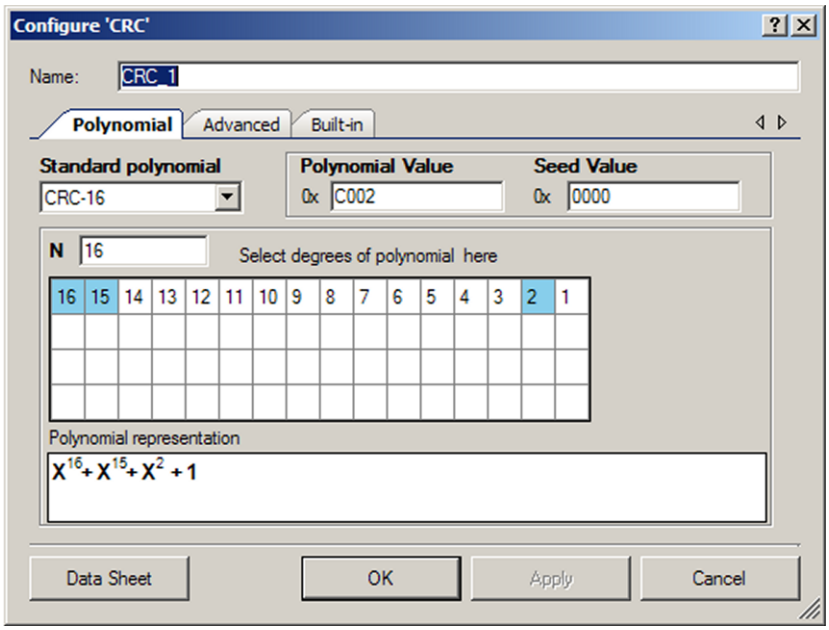
di – 入力

CRC の計算に使用される、シリアル ビットストリームを提供するデータ入力。

コンポーネント・パラメータ

CRC コンポーネントをデザイン上にドラッグし、ダブルクリックして **[Configure (設定)]**ダイアログを開きます。このダイアログには、CRC コンポーネントのセットアップをガイドする複数のタブがあります。

[Polynomial (多項式)] タブ



Standard Polynomial (標準多項式)

[Standard polynomial (標準多項式)] コンボボックスに提供されている標準の CRC 多項式のいずれかを選択するか、カスタム多項式を生成できます。各標準多項式に関するその他の情報は、ツールのヒントに表示されます。デフォルトは **CRC-16** です。

多項式名	多項式	用途
カスタム	ユーザ定義	汎用
CRC-1	x + 1	パリティ
CRC-4-ITU	$x^4 + x + 1$	ITU G.704
CRC-5-ITU	$x^5 + x^4 + x^2 + 1$	ITU G.704
CRC-5-USB	$x^5 + x^2 + 1$	USB
CRC-6-ITU	$x^6 + x + 1$	ITU G.704
CRC-7	$x^7 + x^3 + 1$	電話通信システム、MMC
CRC-8-ATM	$x^8 + x^2 + x + 1$	ATM HEC
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$	1 ワイヤ バス

多項式名	多項式	用途
CRC-8-Maxim	$x^8 + x^5 + x^4 + 1$	1 ワイヤ バス
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	汎用
CRC-8-SAE	$x^8 + x^4 + x^3 + x^2 + 1$	SAE J1850
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	汎用
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	電話通信システム
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	CAN
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$	XMODEM、X.25、V.41、Bluetooth、PPP、IrDA、CRC-CCITT
CRC-16	$x^{16} + x^{15} + x^2 + 1$	USB
CRC-24-Radix64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$	汎用
CRC-32-IEEE802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	Ethernet、MPEG2
CRC-32C	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{+x}9^{+x}8^{+x}6^{+1}$	汎用
CRC-32K	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^{+x}4^{+x}2^{+x+1}$	汎用
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$	ISO 3309
CRC-64-ECMA	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{+x}35^{+x}33^{+x}32^{+x}31^{+x}29^{+x}27^{+x}24^{+x}23^{+x}22^{+x}21^{+x}19^{+x}17^{+x}13^{+x}12^{+x}10^{+x}9^{+x}7^{+x}4^{+x+1}$	ECMA-182

Polynomial Value (多項式値)

このパラメーターは 16 進数で表現されています。標準多項式のいずれかが選択されている場合は、自動的に計算されます。手動で入力することもできます (「カスタム多項式」を参照)。

シード値

このパラメーターは 16 進数で表現されています。可能な最大値は 2^N-1 です。

N

このパラメーターは多項式の全次数を定義します。可能な値は 1～64 ビットです。で表での数字は、多項式中にどの次数が含まれているのか示しています。選択した数のセルは青、その他は白になっています。アクティブなセルの数は N に等しくなります。数は、降順に配列されています。セルをクリックし、数値を選択または選択解除できます。

Polynomial Representation (多項式表現)

このパラメーターは数学的な式として、多項式の結果を表示します。

Custom Polynomials (カスタム多項式)

3 つの異なる方法で、カスタム多項式を入力できます。

標準の多項式に多少の変更を加える

- 標準多項式のいずれかを選択します。
- 該当するセルをクリックし、表で必要な次数を選択します。**[Standard Polynomial (標準多項式)]** のテキストが **[Custom (カスタム)]** に変わります。
- 多項式値が、表示された多項式に基づいて自動的に再計算されます。

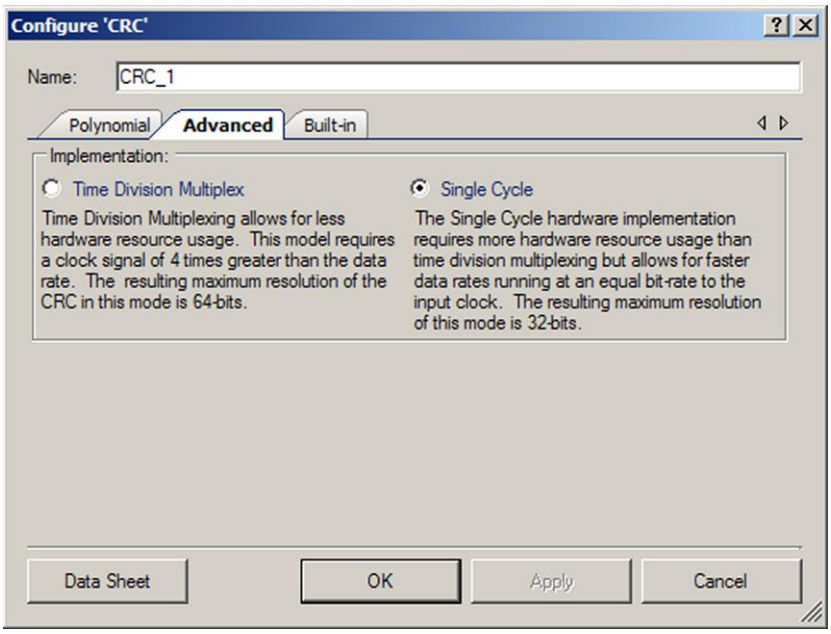
Use Polynomial Degrees (多項式の次数を使用)

- **N** テキストボックスにカスタム多項式を入力します。**[Standard Polynomial (標準多項式)]** のテキストが **[Custom (カスタム)]** に代わります。
- 該当するセルをクリックして、表で必要な次数を選択します。
- **[Polynomial Representation (多項式)]** で多項式を確認します。
- 多項式値が、表示された多項式に基づいて自動的に再計算されます。

Use Hexadecimal Format (16 進法形式の使用)

- **Polynomial Value** テキスト ボックスに、16 進法形式で多項式値を入力します。
- **[Enter]** を押すか、別のコントロールに切り替えます。**[Standard Polynomial (標準多項式)]** が **[Custom (カスタム)]** に変わります。
- N 値と多項式の次数は、入力した多項式値を基にして再計算されます。

Advanced タブ



Implementation (実装)

このパラメーターは、CRC コンポーネントの実装を定義します。**時分割多重化**または**シングル サイクル**。デフォルトは、**シングル サイクル**です。

ローカル パラメータ (API での使用)

これらのパラメータは、API によって使用され、GUI には表示されません。

- **PolyValueLower (uint32) –** 16 進法形式による多項式値の下位半分が含まれます。デフォルトの分解能は 8 であるため、デフォルトは 0xB8h (LFSR = [8,6,5,4]) です。
- **PolyValueUpper (uint32) –** 16 進法形式による多項式値の上位半分が含まれます。デフォルトの分解能は 8 であるため、デフォルトは 0x00h です。
- **SeedValueLower (uint32) –** 16 進法形式によるシード値の下位半分が含まれます。デフォルトの分解能は 8 であるため、デフォルトは 0xFFh です。
- **SeedValueUpper (uint32) –** 16 進法形式によるシード値の上位半分が含まれます。デフォルトの分解能は 8 であるため、デフォルトは 0 です。

クロック選択

このコンポーネントには、内部クロックはありません。クロックソースを必ず取りつけてください。

注 Impremention パラメータで **Time Division Multiplex** を選択した場合、8 を超える分解能で正しい CRC シーケンスを生成するには、データ レートの 4 倍のクロック信号が必要です。

配置

CRC 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

シングル サイクル実装

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	Control/ Count7 セル	フラッシュ	RAM	
1～8 ビット分解能	1	1	1	166	2	4
9～16 ビット分解能	2	1	1	210	2	4
17～24 ビット分解能	3	1	1	287	2	4
25～32 ビット分解能	4	1	1	288	2	4

時分割多重化の実装

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	Control/ Count7 セル	フラッシュ	RAM	
9～16 ビット分解能	1	3	1	242	2	4
17～24 ビット分解能	2	3	1	538	2	4
25～32 ビット分解能	2	3	1	615	2	4
33～40 ビット分解能	3	3	1	763	2	4
41～48 ビット分解能	3	3	1	894	2	4
49～56 ビット分解能	4	3	1	999	2	4
57～64 ビット分解能	4	3	1	1101	2	4

アプリケーション プログラミング インタフェース

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

初期設定で、PSoC Creator により、インスタンス名「CRC_1」が、所定のデザイン中の最初のコンポーネント インスタンスに割り当てられます。コンポーネントのインスタンス名称は、識別子の文法ルールに従って独自の名前に変更できます。インスタンス名は、すべてのグローバル関数名、変数名、定数名のプリフィックスになります。分かりやすいよう、次の表では、インスタンス名「CRC」を使用しています。

関数	説明
CRC_Start()	初期値で、シード値及び多項式レジスタを初期化します。CRC の計算は、入力クロックの立ち上がりエッジで開始されます。
CRC_Stop()	CRC の計算を停止します。
CRC_Wakeup()	CRC 設定を復元し、入力クロックの立ち上がりエッジで CRC の計算を開始します。
CRC_Sleep()	CRC の計算を停止し、CRC の設定を保存します。
CRC_Init()	シード値および多項式レジスタを初期値で初期化します。
CRC_Enable()	入力クロックの立ち上がりエッジで CRC の計算を開始します。
CRC_SaveConfig()	シード値及び多項式レジスタを保存します。
CRC_RestoreConfig()	シード値及び多項式レジスタを復元します。
CRC_WriteSeed()	シード値を書き込みます。
CRC_WriteSeedUpper()	シード値の上位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。
CRC_WriteSeedLower()	シード値の下位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。
CRC_ReadCRC()	CRC 値を読み取ります。
CRC_ReadCRCUpper()	CRC 値の上位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。
CRC_ReadCRCLower()	CRC 値の下位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。
CRC_WritePolynomial()	CRC 多項式値を書き込みます。
CRC_WritePolynomialUpper()	CRC 多項式値の上位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。
CRC_WritePolynomialLower()	CRC 多項式値の下位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。
CRC_ReadPolynomial()	CRC 多項式値を読み取ります。
CRC_ReadPolynomialUpper()	CRC 多項式値の上位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

関数	説明
CRC_ReadPolynomialLower()	CRC 多項式値の下位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

グローバル変数

変数	説明
CRC_initVar	CRC が初期化されたかどうかを示します。変数は、0 に初期化され、最初に CRC_Start() が呼び出されると 1 に設定されます。これで、CRC_Start() ルーチンを最初に呼び出した後で、再初期化することなく、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、CRC_Int() 関数を CRC_Start() または CRC_Enable() 関数の前に呼び出すことができます。

void CRC_Start(void)

説明: 初期値で、シード値及び多項式レジスタを初期化します。CRC の計算は、入力クロックの立ち上がりエッジで開始されます。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_Stop(void)

説明: CRC の計算を停止します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_ Sleep(void)

説明: CRC の計算を停止し、CRC の設定を保存します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_ Wakeup(void)

説明: CRC 構成を復元し、入力クロックの立ち上がりエッジで、CRC の計算を開始します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_ Init(void)

説明: 初期値で、シード値及び多項式レジスタを初期化します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_ Enable(void)

説明: 入力クロックの立ち上がりエッジで、CRC の計算を開始します。

パラメータ: なし

返り値: なし

副作用: なし

void LCD_Char_SaveConfig(void)

説明: 初期のシード値及び多項式レジスタを保存します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_ RestoreConfig (void)

説明: 初期のシード値及び多項式レジスタを復元します。

パラメータ: なし

返り値: なし

副作用: なし

void CRC_WriteSeed(uint8/16/32 seed)

説明: シード値を書き込みます。

パラメータ: uint8/16/32 シード: シード値

返り値: なし

副作用: シード値は、マスク = $2^{Resolution}$ - 1 に準じてカットされます。

例えば、CRC 分解能が 14 ビットの場合、マスク値が次のようになります。
マスク = 2^{14} – 1 = 0x3FFFu。

シード値 = 0xFFFFu はカットされ、シードおよびマスク = 0xFFFFu および 0x3FFFu = 0x3FFFu となります。

void CRC_WriteSeedUpper(uint32 seed)

説明: シード値の上位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: uint32 seed: シード値の上位半分

返り値: なし

副作用: シード値の上位半分は、マスク = $2^{Resolution - 32}$ - 1 に準じてカットされます。

例えば、CRC 分解能が 35 ビットの場合は、マスク値が次のようになります。

$2^{(35 - 32)}$ – 1 = 2^3 – 1 = 0x0000 0007u。

シード値の上位半分 = 0x0000 00FFu はカットされ、

シードの上位半分およびマスク = 0x0000 00FFu および 0x0000 0007u = 0x0000 0007uになります。

void CRC_WriteSeedLower(uint32 seed)

説明: シード値の下位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: uint32 seed: シード値の下位半分

返り値: なし

副作用: なし

uint8/16/32 CRC_ReadCRC(void)

説明: CRC 値を読み取ります。

パラメータ: なし

返り値: uint8/16/32: CRC 値を返します

副作用: なし

uint32 CRC_ReadCRCUpper(void)

説明: CRC 値の上位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: なし

返り値: uint32: CRC 値の上位半分を返します

副作用: なし

uint32 CRC_ReadCRCLower(void)

説明: CRC 値の下位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: なし

返り値: uint32: CRC 値の下位半分を返します

副作用: なし

void CRC_ WritePolynomial(uint8/16/32 polynomial)

説明: CRC 多項式値を書き込みます。

パラメータ: uint8/16/32 多項式: CRC 多項式

返り値: なし

副作用: 多項式値は、マスク = $2^{Resolution}$ - 1 に準じてカットされます。例えば、CRC 分解能が 14 ビットの場合、マスク値が次のようになります。マスク = 2^{14} – 1 = 0x3FFFu。

多項式値 = 0xFFFFu はカットされ、

多項式値およびマスク = 0xFFFFu および 0x3FFFu = 0x3FFFu。

void CRC_WritePolynomialUpper(uint32 polynomial)

説明: CRC 多項式値の上位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: uint32 多項式: CRC 多項式値の上位半分

返り値: なし

副作用: 多項式値の上位半分は、マスク = $2^{(Resolution - 32)} - 1$ に準じてカットされます。例えば、CRC 分解能が 35 ビットの場合は、マスク値が次のようになります。

$2^{(35 - 32)} – 1 = 2^3 – 1$ = 0x0000 0007u.

多項式値の上位半分 = 0x0000 00FFu はカットされ、

多項式の上位およびマスク = 0x0000 00FFu および 0x0000 0007u = 0x0000 0007u になります。

void CRC_WritePolynomialLower(uint32 polynomial)

説明: CRC 多項式値の下位半分を書き込みます。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: uint32 多項式: CRC 多項式値の下位半分

返り値: なし

副作用: なし

uint8/16/32 CRC_ ReadPolynomial(void)

説明: CRC 多項式値を読み取ります。

パラメータ: なし

返り値: uint8/16/32: CRC 多項式値を返します

副作用: なし

uint8/16/32 CRC_ ReadPolynomialUpper(void)

説明: CRC 多項式値の上位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: なし

返り値: uint32: CRC 多項式値の上位半分を返します

副作用: なし

uint32 CRC_ ReadPolynomialLower (void)

説明: CRC 多項式値の下位半分を読み取ります。33～64 ビットの CRC に対してのみ生成されます。

パラメータ: なし

返り値: uint32: CRC 多項式値の下位半分を返します。

副作用: なし

ファームウェア・ソースコードのサンプル

PSoC Creator は、[Find Example Project（サンプルプロジェクトを検索）] ダイアログに多数のサンプルプロジェクトを提供しており、そこには回路図およびサンプル コードが含まれています。コンポーネント固有の例を見るには、[Component Catalog] または回路図に置いたコンポーネント インスタンスからダイアログを開きます。一般例については、[Start Page] または **[File（ファイル）]** メニューからダイアログを開きます。必要に応じてダイアログにある **Filter Options** 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（サンプルプロジェクトを検索）」を参照してください。

機能説明

CRC は、リニア フィードバック シフト レジスタ (LFSR) として実装されます。シフト レジスタは、LFSR 関数を計算します。多項式レジスタは、LFSR 多項式を定義する多項式を保持し、シード レジスタは、開始データの初期化をイネーブルにします。

シード値及び多項式レジスタは、コンポーネントを開始する前に初期化する必要があります。

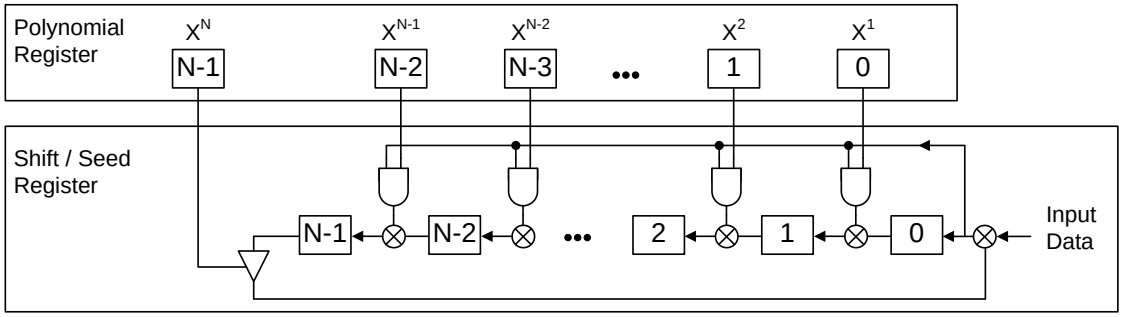
N ビット LFSR 結果の計算は、$X^0 = 1$ となる X^0 項 が最後の項となる N+1 項の多項式によって指定されます。例えば、広く利用されている CRC–CCITT 16 ビットの多項式は、$X^{16}+X^{12} + X^5 +1$ です。CRC アルゴリズムは、X^0 項が存在すると想定するため、N ビット結果の多項式は、N+1 ビットの仕様ではなく、N ビットによって表現できます。

多項仕様を指定するには、各項の存在を 1 で表す、完全な多項式に該当する N+1 ビットの 2 進数を書き込みます。CRC–CCITT 多項式値は、10001000000100001b のようになります。次に、一番右のビット (X^0 項) をドロップし、CRC 多項式値を取得します。CRC–CCITT 例を実装するため、多項式レジスタに値 8810h がロードされます。

入力クロックの立ち上がりエッジにより、入力データ ストリームの各ビットがシフトされます。これは、指定された CRC アルゴリズムを計算するシフト レジスタを通して、MSB から始まります。入力データの各バイトの CRC を計算するには、8 クロックが必要です。

初期シード値が失われることに注意してください。シード値は、シフト レジスタを各データセットにつき一度だけ初期化するためにのみ使用されるため、初期シード値が失われることで悪影響が出ることはありません。

ブロックダイアグラムと設定



タイミング図

図 1. 時分割多重化の実装モード

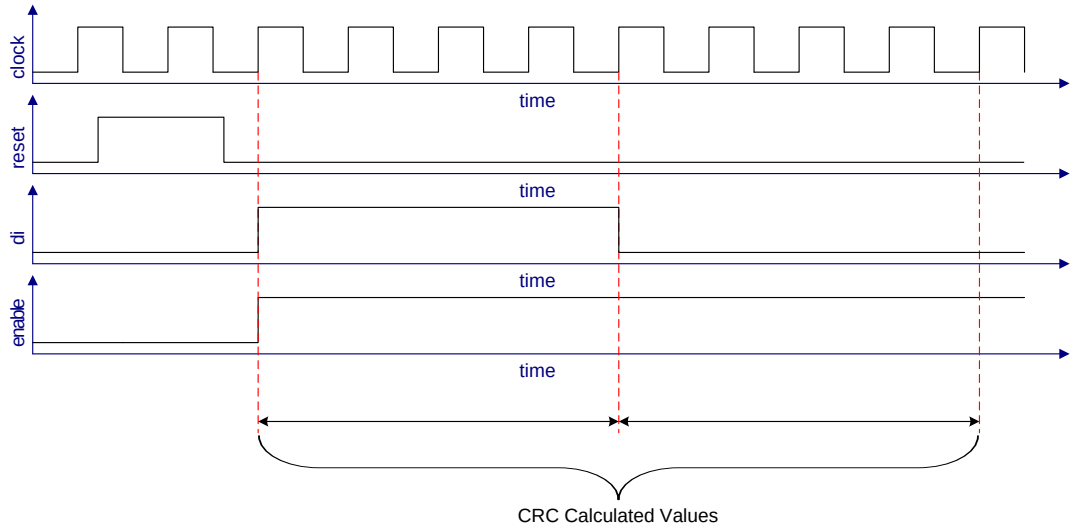
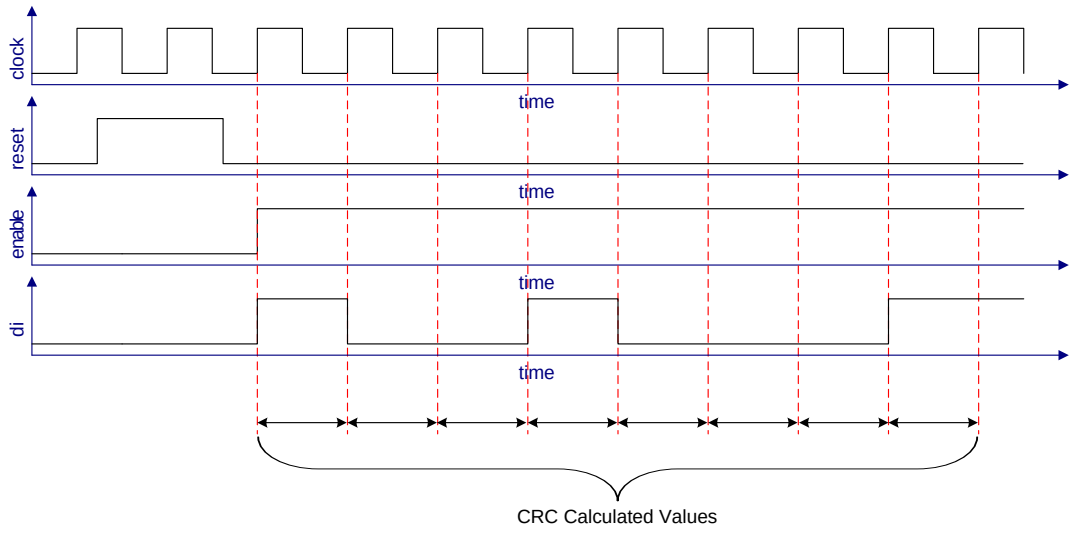


図 2. シングル サイクル実装モード



DC 電気的特性と 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。

「公称ルーティングでの最大」タイミング特性

パラメータ	説明	構成[1]	Min	Typ	Max	単位
f_{CLOCK}	コンポーネントクロック周波数 [2]	構成 1			45	MHz
		構成 2			30	MHz
		構成 3			41	MHz
		構成 4			24	MHz
		構成 5			35	MHz
		構成 6			21	MHz
t_{CLOCKH}	入力クロック HIGH 時間 [3]	該当せず		0.5		$1/f_{CLOCK}$
t_{CLOCKL}	入力クロック LOW 時間 [3]	該当せず		0.5		$1/f_{CLOCK}$
Inputs (入力)						
t_{PD_ps}	入力パス遅延、同期ピン [4]	1			STA [5]	ns

[1] 設定:

設定 1:
分解能: 8 ビット
実装: シングル サイクル

設定 2:
分解能: 16 ビット
実装: シングル サイクル

設定 3:
分解能: 16 ビット
実装: 時分割多重化

設定 4:
分解能: 32 ビット
実装: シングル サイクル

設定 5:
分解能: 32 ビット
実装: 時分割多重化

設定 6:
分解能: 64 ビット
実装: 時分割多重化

[2] Time Division Multiplex Implementation (時分割多重化の実装) が選択されると、コンポーネント クロック周波数はデータレートの 4 倍でなければなりません。

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

[4] t_{PD_ps} は、後述されるように静的タイミング (STA) の結果内にあります。ここに挙げた数字は、多くの入力の STA 分析に基づく公称値です。

[5] t_{PD_ps} と t_{PD_si} はルートパスの遅延です。ルーティング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の周波数に直接影響することがあります。これらの値は、Static Timing Analysis Results (静的タイミング分析の結果) に記載されています。

パラメータ	説明	構成[1]	Min	Typ	Max	単位
t_{PD_ps}	入力パス遅延、同期ピン [6]	2			8.5	ns
t_{PD_si}	Sync出力から入力パスへの遅延 (配線)	1,2,3,4			STA [5]	ns
t_{I_clk}	clockXとクロックのアライメント	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パス遅延 (エッジセンシティブ入力)	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パス遅延 (エッジセンシティブ入力)	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

[6] t_{PD_ps} 構成 2 で、デバイスのピン毎に定義された固定値。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。

[7] t_{CY_clock} = 4 x [1/f_{clock}] Time Division Multiplex Implementation (時分割多重化の実装) が選択された場合。

「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	構成 [8]	Min	Typ	Max [9]	単位
f_{CLOCK}	コンポーネントクロック周波数 [10]	構成 1			23	MHz
		構成 2			15	MHz
		構成 3			21	MHz
		構成 4			12	MHz
		構成 5			18	MHz
		構成 6			11	MHz
T_{CLOCKH}	入力クロック HIGH 時間 [11]	該当せず		0.5		$1/f_{CLOCK}$
T_{CLOCKL}	入力クロック LOW 時間 [11]	該当せず		0.5		$1/f_{CLOCK}$
Inputs (入力)						
t_{PD_ps}	入力パス遅延、同期ピン [12]	1			STA [13]	ns

[8]設定:

設定 1:
分解能: 8 ビット
実装: シングル サイクル

設定 2:
分解能: 16 ビット
実装: シングル サイクル

設定 3:
分解能: 16 ビット
実装: 時分割多重化

設定 4:
分解能: 32 ビット
実装: シングル サイクル

設定 5:
分解能: 32 ビット
実装: 時分割多重化

設定 6:
分解能: 64 ビット
実装: 時分割多重化

[9]すべてのルーティングでの最大タイミング数は、公称ルーティング タイミング数を 2 倍に下げて計算されます。コンポーネント インスタンスがこれらの速度以下で動作する場合、このコンポーネントに対してタイミングを気にする必要はありません。

[10] Time Division Multiplex Implementation (時分割多重化の実装) が選択されると、コンポーネント クロック周波数はデータレートの 4 倍でなければなりません。

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

[12] 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。ここに挙げた数字は、多くの入力の STA 分析に基づく公称値です。

パラメータ	説明	構成 [8]	Min	Typ	Max [9]	単位
t_{PD_ps}	ピンからSyncへの入力パス遅延 [14]	2			8.5	ns
t_{PD_si}	Sync出力から入力パスの遅延(配線)	1,2,3,4			STA [5]	ns
t_{I_clk}	clockXとクロックのアライメント	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パス遅延 (エッジセンシティブ入力)	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パス遅延 (エッジセンシティブ入力)	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} [15]			ns
t_{IL}	入力 LOW 時間	1,2,3,4	t_{CY_clock} [15]			ns

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 結果を用いた設計の場合、以下の手法で最大値を計算します。

f_{CLOCK} 最大コンポーネントクロック周波数が、名前付きの外付けクロックとして、クロック サマリのタイミング結果に表示されます。下図は、*_timing.html* による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	

入力パス遅延とパルス幅

入力の機能を特性化する場合は、どのように構成しても、すべての入力は、図 3 に示されるように、4 つの可能な設定のいずれかになります。

---

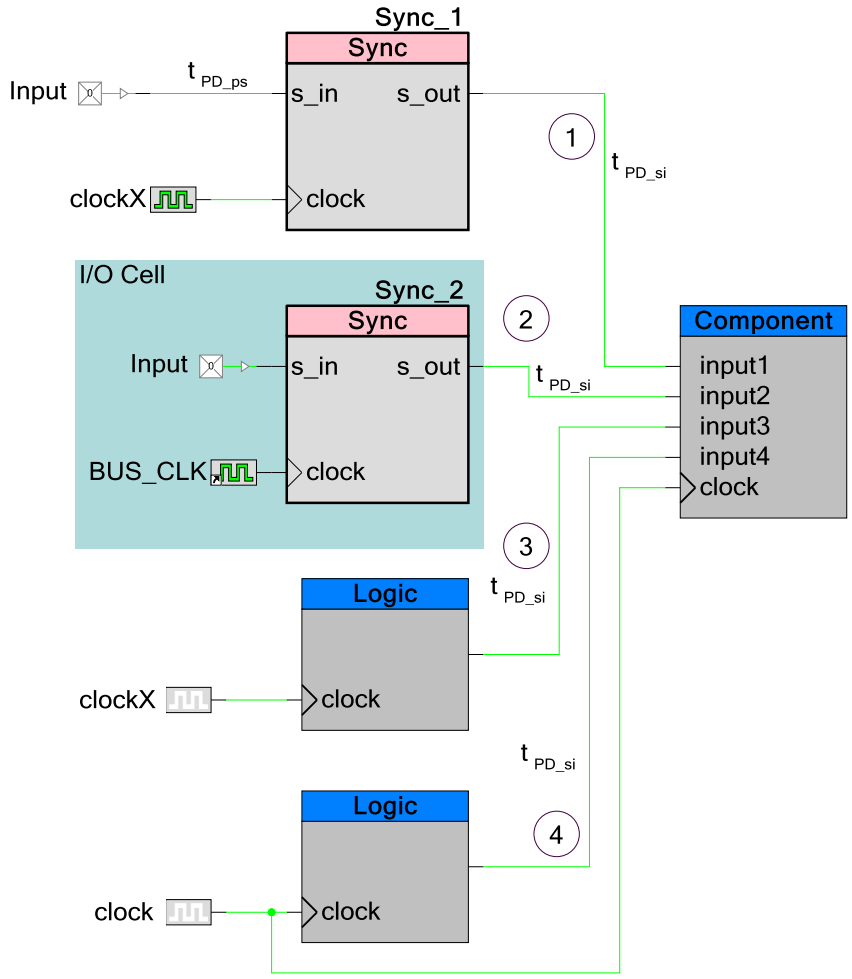
[13] t_{PD_ps} と t_{PD_si} はルートパスの遅延です。ルーティングは動的なためこれらの値は変化することがあり、最大コンポーネント クロックと同期クロックの周波数に直接影響することがあります。これらの値は、Static Timing Analysis Results (静的タイミング分析の結果) に記載されています。

[14] t_{PD_ps} 構成 2 で、デバイスのピン毎に定義された固定値。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。

[15] t_{CY_clock} = 4 x [1/f_{clock}] Time Division Multiplex Implementation (時分割多重化の実装) が選択された場合。

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、各入力でどの入力構成を設定したか、また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 (静的タイミング分析、STA) の結果を使用して、システムの特性分析を行う方法について説明します。

図 3. 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 8
1	クロック	master_clock	図 6
1	クロック	clockX = clock [16]	図 4

[16] クロック周波数は同等ですが、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
1	クロック	clockX > clock	図 5
1	クロック	clockX < clock	図 7
2	master_clock	master_clock	図 8
2	クロック	master_clock	図 6
3	master_clock	master_clock	図13
3	クロック	master_clock	図 11
3	クロック	clockX = clock [16]	図 9
3	クロック	clockX > clock	図10
3	クロック	clockX < clock	図 12
4	master_clock	master_clock	図13
4	クロック	クロック	図 9

1. 入力は、デバイスピンによって駆動され、内部で「sync」コンポーネントと同期します。このコンポーネントには、このコンポーネント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が供給されます (すべての内部クロックは master_clock から派生)。

   この方法で設定される入力を特徴付けた場合、clockX は、コンポーネント クロックより高速、低速、または同じのいずれでもかまいません。また、図 4、図 5、図 7、図 8 に示されているように、特性化パラメータを生成する、master_clock にも等しくなる場合があります。

2. この入力は、デバイスピンによって駆動され、master_clock を使用してそのピンと同期されます。

   このような方法で構成された入力の特性を分析する際は、master_clock はコンポーネントのクロック以上の速度です（遅いことはありません）。これにより、図 5 および図 8 に示されているように、特性化パラメータが生成されます。

図 4. 入力構成 1 および 2。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 (クロックおよび clockX のエッジ アライメントは保証されません)

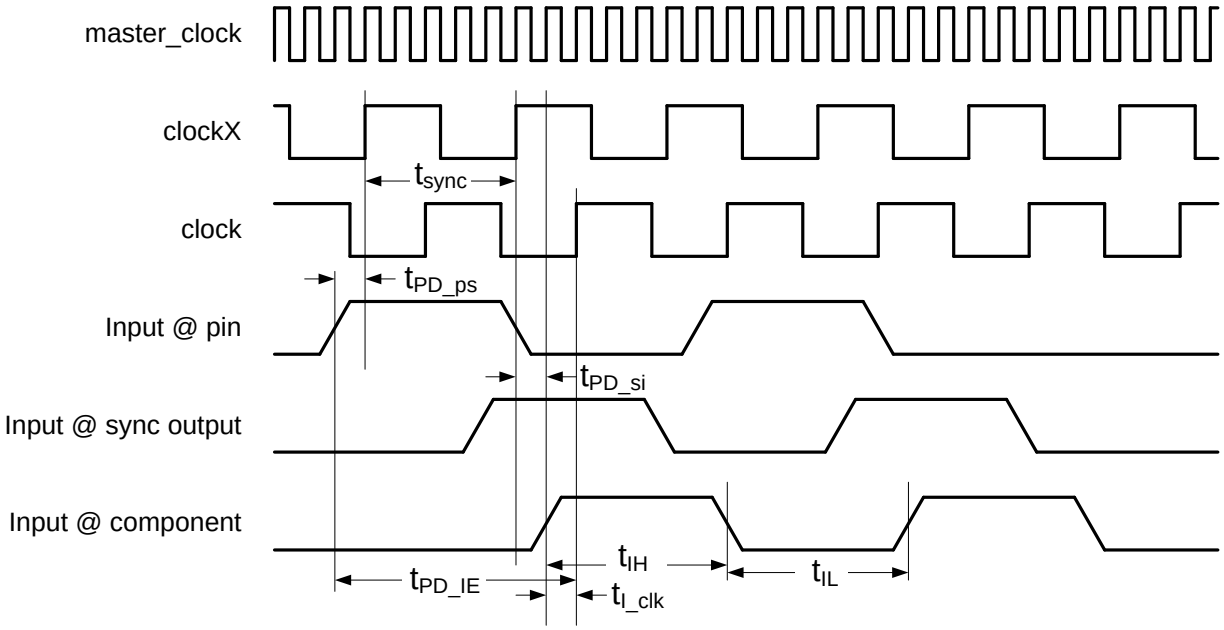


図 5. 入力構成 1 および 2。 シンクロナイザクロック周波数＞コンポーネントクロック周波数

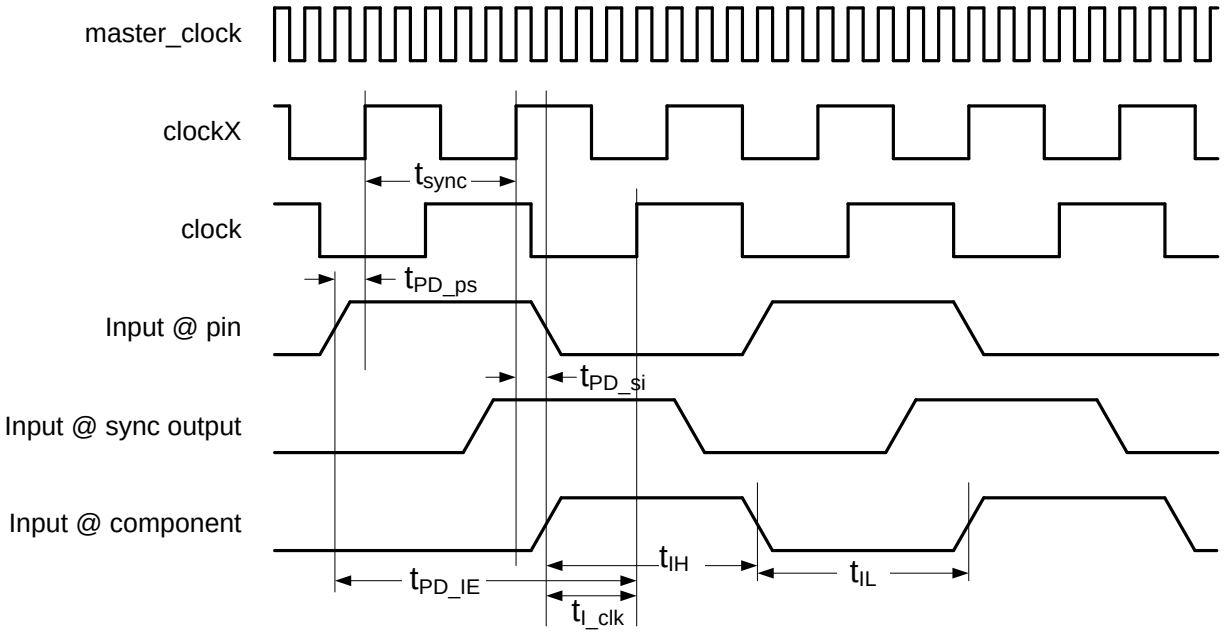


図 6. 入力構成 1 および 2。[シンクロナイザクロック周波数 == master_clock ＞ コンポーネントクロック周波数

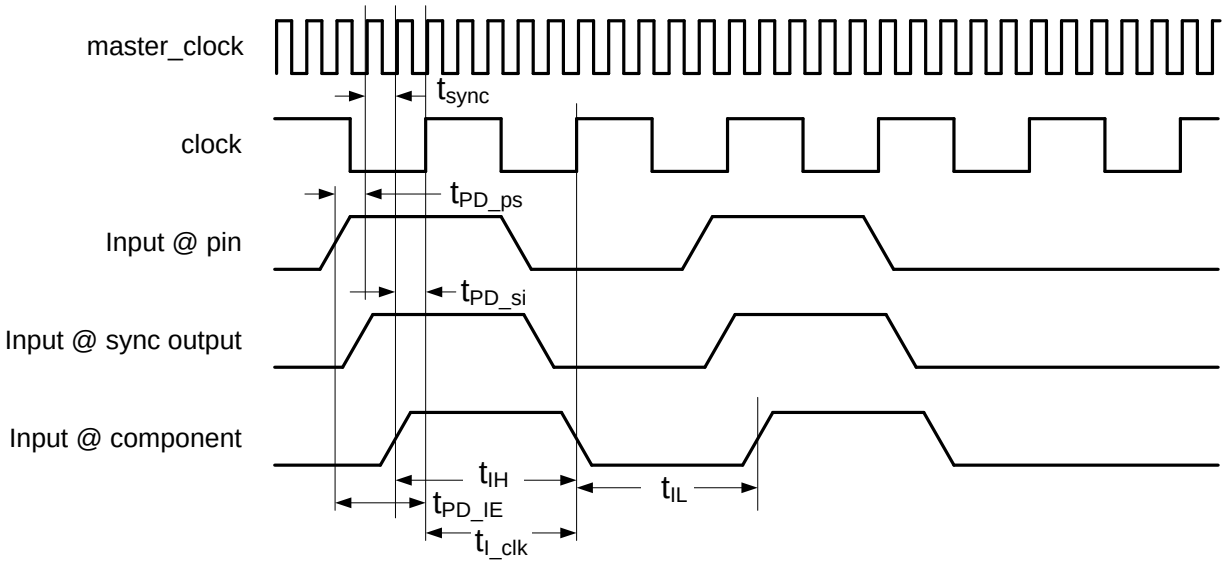


図 7. 入力構成 1。シンクロナイザクロック周波数 ＜ コンポーネントクロック周波数

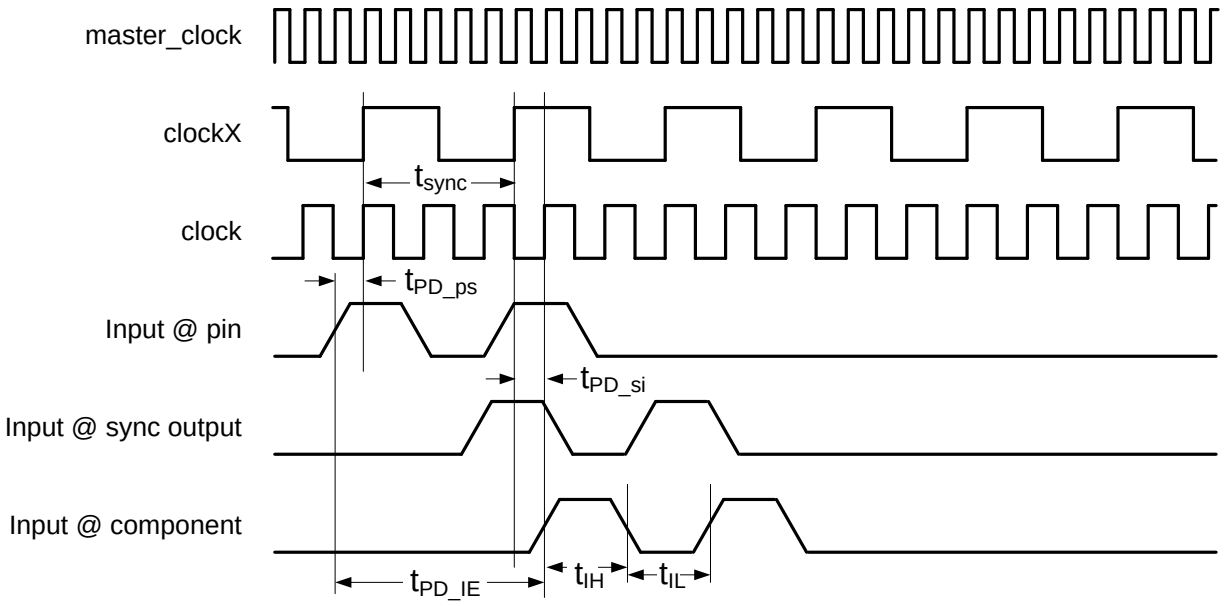
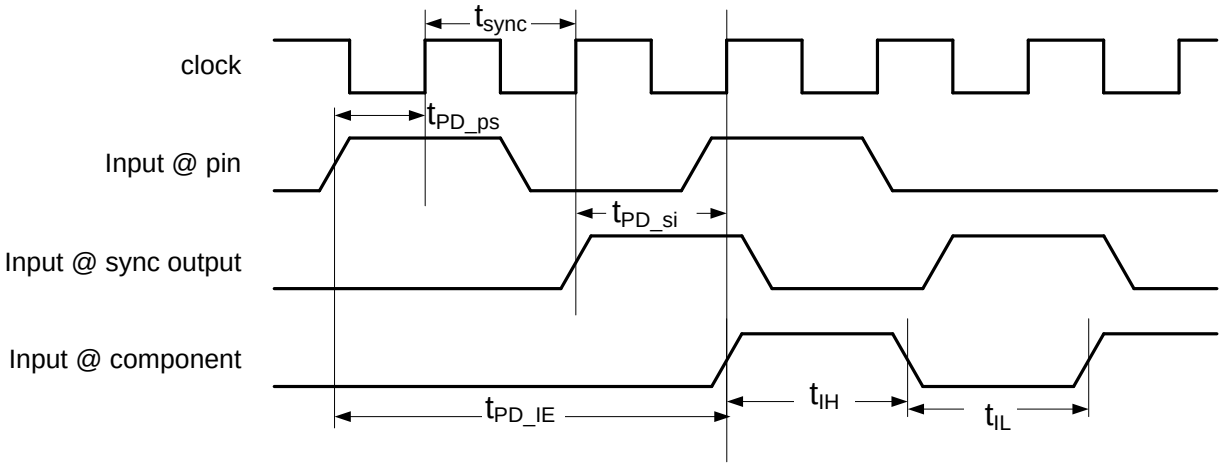


図 8. 入力構成 1 および 2。シンクロナイザクロック==コンポーネントクロック==master_clock



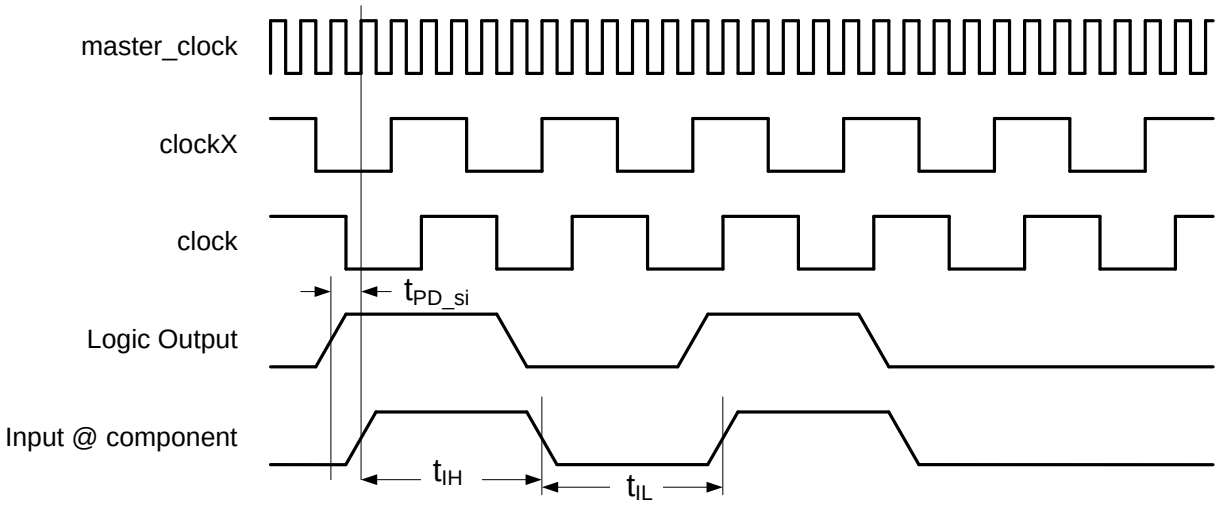
3. 入力は PSoC 内部のロジックに駆動されます。これはコンポーネントが使用するクロックとは異なるクロックをベースにして同期しています（すべての内蔵クロックは master_clock によって駆動しています）。

   この方法で設定された入力を特性化する場合、シンクリナイザ クロックは、図 9、図 10、図 12 に示されている特性化パラメータを生成するコンポーネント クロックより高速、低速、または等しくなります。

4. 入力は PSoC 内部のロジックに駆動されます。これはコンポーネント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。

   この方法で設定された入力を特性化する場合、シンクロナイザ クロックは、図 13 に示されている特性化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 9. 入力構成 3 のみ。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（クロックおよび clockX のエッジ アライメントは保証されません）



この数字は、静的タイミング分析のクロックに対する理解を示します。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、立ち上がりエッジでアライメントされないことがあります。そのため、静的タイミング分析ツールには、クロックが同期しているエッジがどちらか判別できず、最低の 1 master_clock サイクルを想定する必要があります。これは、t_{PD_si} がシステムの master_clock に与える影響が限定的であることを意味します。このパスの遅延が長すぎると、master_clock セットアップ時間の違反が表示されます。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 10. 入力構成 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

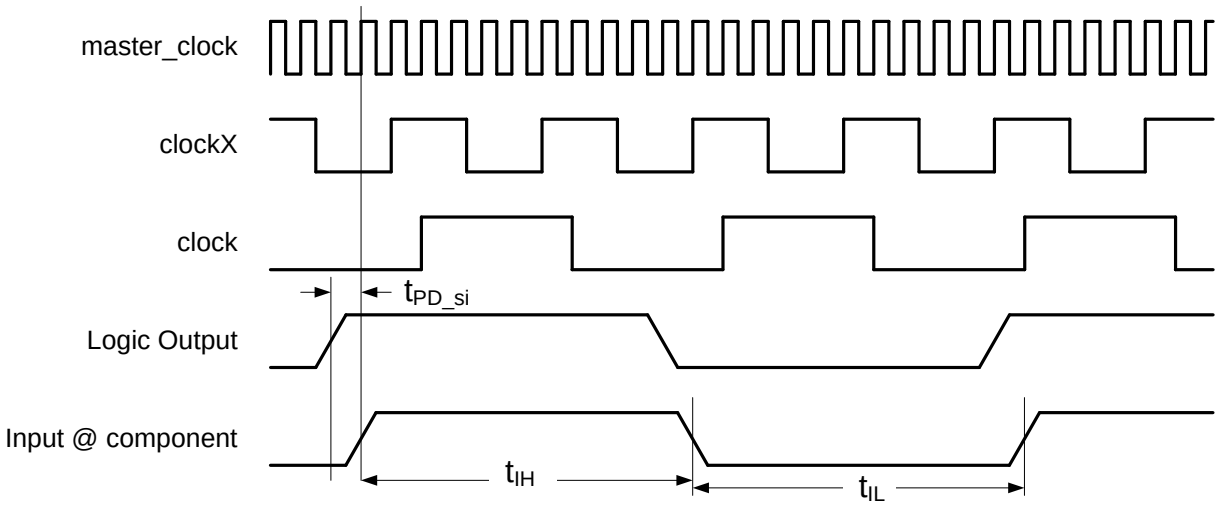


図 9 に示されているのとほぼ同じ方法で、すべてのクロックは、master_clock から生成されます。STA は、この構成で 1master_clock サイクル分、master_clock における t_{PD_si} の制限を示します。このパスの遅延が長すぎると、master_clock セットアップ時間の違反が発生します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 11. 入力構成 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

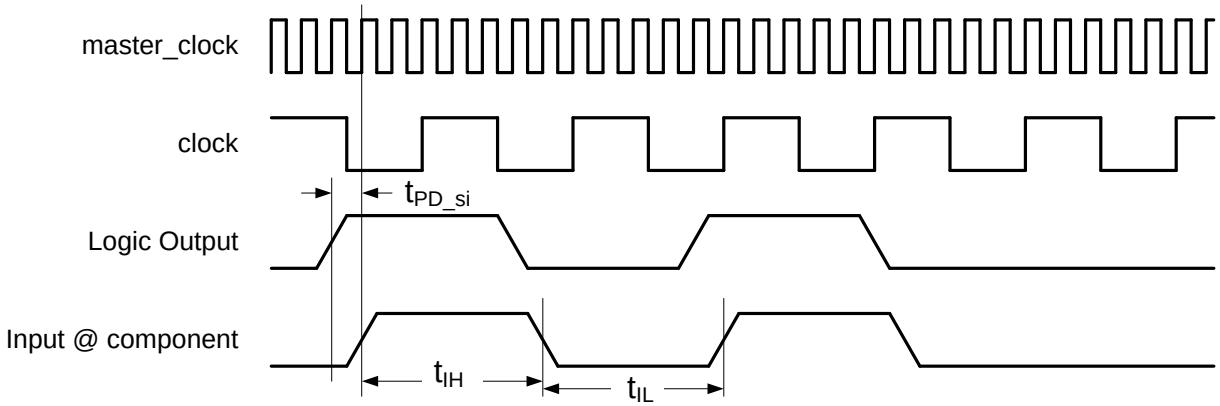


図 12. 入力構成 3。シンクロナイザクロック周波数 ＜ コンポーネントクロック周波数

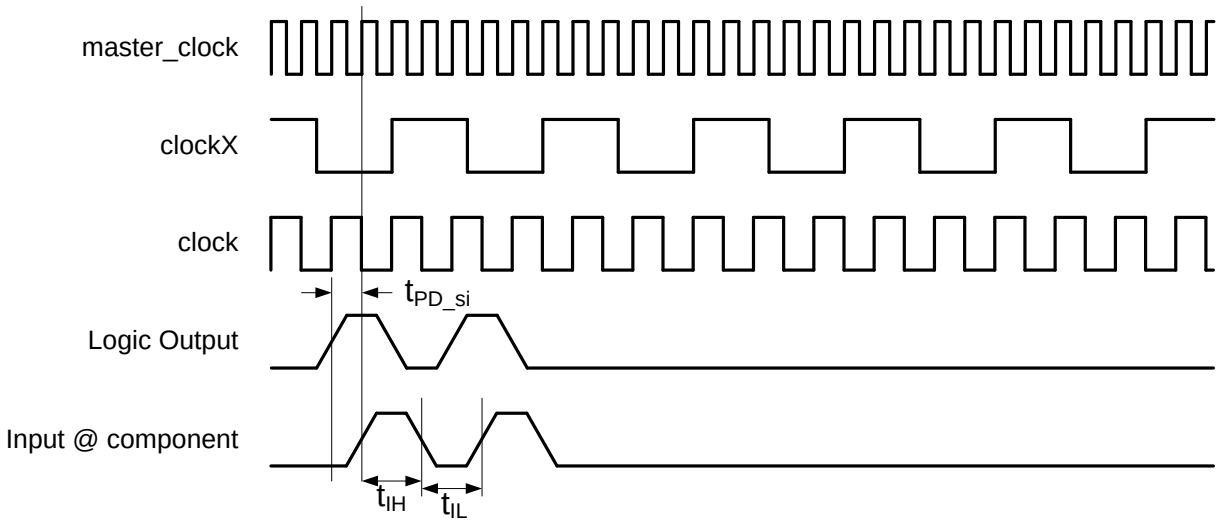
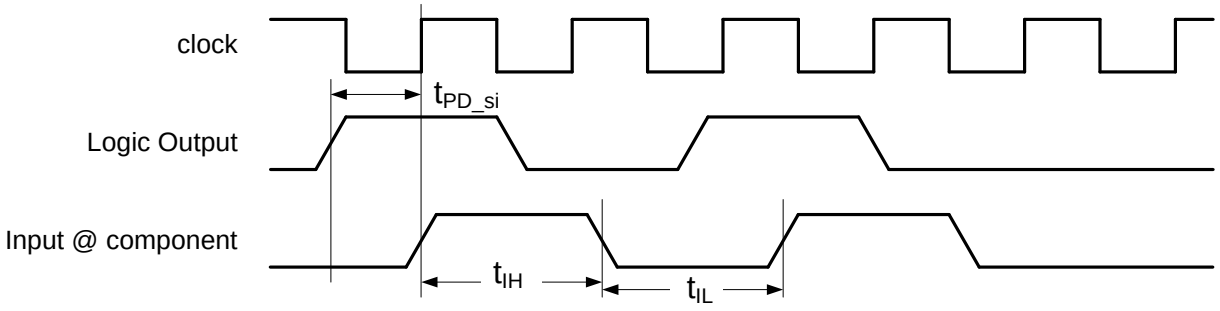


図 9 に示されているのとほぼ同じ方法で、すべてのクロックは master_clock から生成されます。STA は、この構成で 1master_clock サイクル分、master_clock における t_{PD_si} の制限を示します。このパスの遅延が長すぎると、master_clock セットアップ時間の違反が発生します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 13. 入力構成 4 のみ。シンクロナイザクロック == コンポーネントクロック



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}、および t_{I_Clk} で定義されます。非常に重要なことは、t_{PD_si} によって最大コンポーネント クロック周波数が定義されることです。t_{I_Clk} は STA の結果によるものではなく、通常は t_{PD_IE} が登録されたときに示されます。これはシンクロナイザ クロックとコンポーネントクロックの間のルート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} は、*_timing.html* ファイルで定義されている入力設定時間を参照してください。2 つ以上入力のファンアウトがある場合があるので、これらのパスの最大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、レジスタ～レジスタ時間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。2 つ以上パスのファンアウトがある場合があるので、これらのパスのなかで最大のもの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

出力のパス遅延の特性分析を行う場合、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、出力の送信先を知らなければなりません。このコンポーネントでは、すべての出力がコンポーネント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、上述のロジック～入力の説明に記載されているレジスタ～レジスタ時間を見ます（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、*_timing.html* STA 結果のクロック～出力時間を調べます。

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
2.10	実装パラメータに対する、変更されたエラーメッセージとその外観。	
	多項式の次数「N」の設定を 64 ビット解像度に固定しました。	
	固定多項式値の検証。	
2.0.b	データシートのマイナーな編集と更新	
2.0.a	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	
2.0	PSoC ES3 シリコンでのサポートが追加されました。変更は以下のとおりです。 ■ 時分割多重化の実装モードに 4x クロックを追加 ■ 1x クロックでのシングル サイクル実装が 1～32 ビットで使用できるようになりました。 ■ 4x クロックでの時分割多重化の実装モードが 9～64 ビットで使用できるようになりました。 ■ 非同期入力信号のリセットが追加されました。 ■ 同期入力信号のイネーブルが追加されました。 ■ [Implementation (実装)] (時分割多重化、シングル サイクル) パラメータの [Configure (設定)] ダイアログに新しい [Advanced (詳細)] ページが追加されました	PSoC 3 ES3 デバイスをサポートする新しい要求により、CRC コンポーネントの新しい 2.0 バージョンが作成されました。
	CRC_Sleep()/CRC_Wakeup() および CRC_Init()/CRC_Enable() API が追加されました。	低消費電力モードをサポートし、ほとんどのコンポーネントの初期化と有効化の制御を分離する共通インターフェースを提供するため。

バージョン	変更の説明	変更の理由 / 影響
	関数 CRC_WriteSeed() および CRC_WriteSeedUpper() が更新されました。	マスク パラメータは、シード値をカットして、書き込み中に CRC の分解能を定義するために使用されました。
	Resolution (分解能) パラメータへの検証が追加されました。	CRC の分解能は 1～64 ビットです。検証が、入力値を制限するために追加されました。
	リセット DFF のトリガが、次の多項式書き込み関数に追加されました。CRC_WritePolynomial()、CRC_WritePolynomialUpper()、CRC_WritePolynomialLower()。	CRC の計算を開始する前に、DFF トリガを正しい状態に設定する必要があります (多項式の最上位ビットは常に 1)。この条件を満たすため、シードまたは多項式レジスタへの書き込みにより、DFF トリガがリセットされます。
	次のパラメータで、Expression View が見れるよう、[Configure (設定)] ダイアログが更新されました。PolyValueLower、PolyValueUpper、SeedValueLower、SeedValueLower	Expression View は、記号パラメータに直接アクセスするために使用されます。このビューを使うと、必要に応じて、コンポーネント パラメータと外部パラメータを接続できます。
	[Configure (設定)] ダイアログが更新され、さまざまなパラメータにエラー アイコンが追加されました。	テキスト ボックスに不正な値を入力すると、問題を説明するツールのヒントとともに、エラー アイコンが表示されます。これにより、エラー メッセージを別途表示するより、使いやすくなりました。
1.20	API 生成の方法が変更されました。バージョン 1.10 では、API はカスタマイザの設定から生成されていました。1.20 では、API は、他のほとんどのコンポーネントと同様に、.c および .h ファイルによって提供されます。	この変更により、生成された API を表示して、変更を加えることができるようになりました。また、次のビルドで上書きされることはありません。
	シード値及び多項式パラメータが、16 進法形式に変更されました。	変更は、サイプレスの標準に準拠するためのものです。

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。