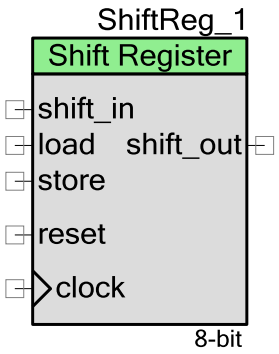
シフトレジスタ(ShiftReg)

2.0

特徴



- 調整可能なシフト レジスタ サイズ: 2 ～ 32 ビット
- 同時シフト インおよびシフト アウト
- 右シフトまたは左シフト
- リセット入力によりシフト レジスタをすべて 0 に強制クリア
- CPU または DMA から読み出し可能なシフト レジスタ値
- CPU または DMA から書き込み可能なシフト レジスタ値

一般的な説明

シフト レジスタ (ShiftReg) コンポーネントは、パラレル レジスタのデータの同期シフト入出力を提供します。パラレル レジスタは CPU または DMA から読み出しまたは書き込みできます。シフト レジスタ コンポーネントは、以下を含む標準の 74xxx シリーズのロジック シフト レジスタと類似した共通の機能を提供します。74164、74165、74166、74194、74299、74595 および 74597。多くのアプリケーションで、シフト レジスタ コンポーネントは、シフトされたビット数を数えるカウンタなどの高レベルのアプリケーション固有の機能を構築するために、ロジックや他のコンポーネントと併用されます。

一般的な使用法では、シフト レジスタは、クロック入力の立ち上がりエッジでデータをシフトする 2 ～ 32 ビット シフト レジスタとして機能します。シフトの方向は設定可能で、入力で MSB がシフト イン、または出力で LSB がシフト アウトされる右シフトと、入力で LSB がシフト インされる、または出力で MSB がシフト アウトされる左シフトが可能です。

シフト レジスタ値は、いつでも CPU または DMA によって書き込み可能です。ロード信号の立ち上がりエッジの後、コンポーネント クロックの立ち上がりエッジで、待機中の FIFO データ (CPU または DMA によって以前に書き込まれた) をシフト レジスタに転送します。オプションのストア入力の立ち上がりエッジが検知されたとき、コンポーネント クロックの立ち上がりエッジで、現在のシフト レジスタ値を FIFO に転送し、これは後で CPU によって読み出すことができます。

シフト レジスタ コンポーネントは、ロード、ストアまたはリセット信号のあらゆる組み合わせで割り込み信号を生成できます。

シフト レジスタの用途

シフト レジスタの最も一般的な用途は、シリアル インタフェースとパラレル インタフェースを変換することです。パラレルでは多くの回路でビットのグループとして操作するため、有用ですが、シリアル インタフェースはよりシンプルに構築できます。

シフト レジスタは、単純な遅延回路としても使用できます。多くの場合、シフト レジスタをお使いのアプリケーションで必要とされる方法で機能させるには、追加のアプリケーション固有の回路が必要です。たとえば、複数のイベントが発生した後にシフトされたデータをストアするカウンタやステートマシンが挙げられます。

シフト レジスタの一般的な用途は、SPI プロトコルの例に見られるように、クロックを基にデータを 8 ビット シフト インまたはシフト アウトすることです。通信プロトコルを構築する場合は、その通信プロトコル用の高レベル コンポーネントがすでに存在しないか確認してください。

入出力接続

このセクションでは、シフト レジスタのためのさまざまな入力および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、シンボルから隠されている可能性があることを示します。

shift_in – Input *

シフト方向に従った、シフト レジスタの MSB または LSB へのシリアル データ入力。**Use Shift In** チェックボックスが選択されている場合に、この端子が表示されます。

load – Input *

ロード入力信号によって、待機中 (CPU または DMA によって FIFO に以前に書き込まれた) の入力 FIFO データのシフト レジスタへの転送がトリガされます。転送は、ロード立ち上がりエッジが検知された後のコンポーネント クロックの立ち上がりエッジで行われます。この端子は、**Use Load** チェックボックスが選択されている場合に表示されます。ロード パルスのデューティ サイクルは任意であることに注意してください。ただし、幅が最低 1 コンポーネント クロック サイクルである必要があります。ロード信号は、他のポジティブ エッジが検知される前に、最低 1 サイクル低い必要があります。

store – Input *

出力 FIFO への現在のシフト レジスタ値の転送は、ストア入力信号によってトリガされます。転送は、ストア信号立ち上がりエッジが検知された後の最初のコンポーネント クロックの立ち上がりエッジで発生します。ロード パルスのデューティ サイクルは任意であることに注意してください。ただし、幅が最低 1 コンポーネント クロック サイクルである必要があります。ストア信号は、次のストアイベント前に最低 1 コンポーネント クロック期間低い必要があります。これで、ShiftReg_ReadData() API ルーチンを使用して FIFO からのデータを読み出せます。この端子は、**Use Store** チェックボックスが選択されている場合に表示されます。

reset – Input

リセット入力 (アクティブ高) によりシフト レジスタ全体が 0 に設定されます。この入力は FIFO の内容には影響を与えません。リセット入力はクロック入力と同期されます。リセット入力は外部接続なしでフローティング状態のままにしてもかまいません。リセット ラインに何も接続されていない場合、コンポーネントによって論理 0 の固定値が割り当てられます。

clock – 入力

コンポーネントのクロック ソース。設定によっては、この信号はクロックではなくイネーブルとして動作します。

shift_out – Output *

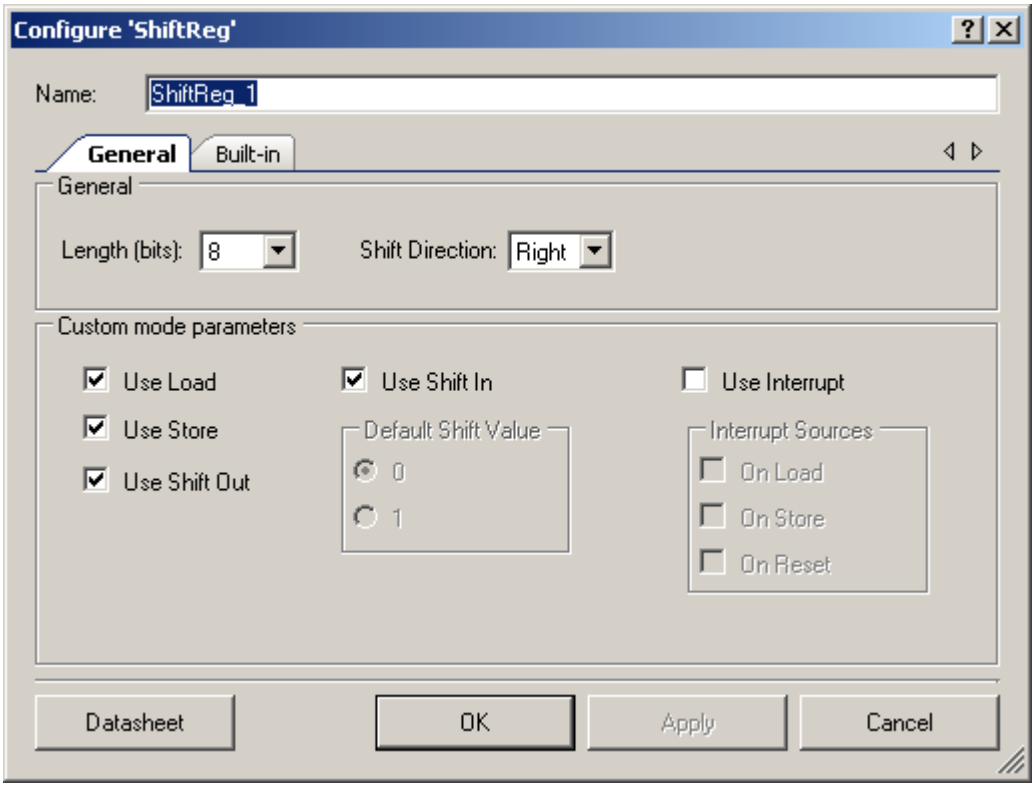
シフトレジスタからのシリアルデータを出力。レジスタの MSB か LSB かはシフト方向による。この端末は、**Use Shift Out** チェックボックスが選択されている場合、表示されます。

interrupt – Output *

割り込み信号はシフト レジスタ コンポーネントによって生成されます。割り込みは指定されたパラメータを基に生成されます。この端末は、**Use Interrupt** チェックボックスが選択されている場合に表示されます。

コンポーネント パラメータ

設計にシフト レジスタ コンポーネントをドラッグしてダブルクリックすると **Configure** ダイアログが開きます。



Length (bits)

このパラメータは、シフト レジスタの長さをビット単位で規定します。有効な値は、2 ～ 32 ビットです。デフォルトは **8** です。

Shift Direction

このパラメータは、**Right** または **Left** のシフト方向を規定します。デフォルトは **Right** (LSB が先) です。

Use Load

このオプションを選択すると、負荷入力端末がシフト レジスタ記号に含まれます。ロード信号は、ロード信号のポジティブ エッジを検知した後のコンポーネント クロックの立ち上がりエッジで、入力 FIFO からのワードが シフト レジスタに転送されるよう、制御ロジックに内部で配線されます。

Use Load を選択した場合、出力 FIFO で動作するための ShiftReg_WriteRegValue()、ShiftReg_ReadRegValue()、ShiftReg_GetIntStatus() API が生成されます。*component.h* ファイルには、必要な API のプロトタイプと #define 定数が含まれています。

Use Load が選択されていない場合、ロード端末はコンポーネント記号に表示されず、関連する API ルーチンは生成されません。

Use Store

このオプションが選択されている場合、ストア入力端末はシフト レジスタ記号に含まれます。ストア信号は、ストア信号立ち上がりエッジを検知した後のコンポーネント クロックの立ち上がりエッジ、シフト レジスタの現在のワードが出力 FIFO に転送されるよう、制御ロジックに内部的に転送されます。

Use Store を選択した場合、出力 FIFO で動作するための ShiftReg_WriteRegValue()、ShiftReg_ReadRegValue()、ShiftReg_GetIntStatus() API が生成されます。*component.h* ファイルには、必要な API のプロトタイプと #define 定数が含まれています。

注意

ShiftReg_ReadRegValue() API ルーチンと **Use Store** 出力 FIFO 機能と併用する場合は注意してください。ShiftReg_ReadRegValue() API 実装は、現在のシフト レジスタ ALU 値を出力 FIFO に転送し、FIFO からそのデータを読み出します。ストア信号を使用して出力 FIFO にキャプチャされたが、アプリケーションによってまだ読み出されていないデータは失われます。

Use Store が選択されていない場合、コンポーネント記号にストア端末は表示されず、関連する API ルーチンは生成されません。

Use Shift Out

このパラメータは、シフトレジスタ記号の shift_out 出力が提供されるか否かを指定します。これはデフォルトで選択されています。

Use Shift In

このパラメータは、シフト レジスタ記号の shift_in 入力が提供されるか否かを指定します。これはデフォルトで選択されます。

Default Shift Value

このパラメータを使用すると、シフト レジスタへの入力のデフォルト値を定義できます。このパラメータは、**Use Shift In** パラメータがチェックされていない場合のみ使用されます。**Default Shift Value** パラメータの有効な値は **0** と **1** です。

Use Interrupt

このパラメータが選択されていると、記号上に割り込み出力端末が表示されます。これにより、シフト レジスタによって生成された割り込みを使用することができます。

Use Interrupt が選択されていないと、割り込み端末がシンボル上に表示されず、関連する API ルーチンが生成されます。

Interrupt Sources

このパラメータは、**Use Interrupt** を選択している場合、イネーブルにされます。割り込み信号を使用して、特定の条件の 1 つが発生したことが示されます。割り込みの生成をイネーブルまたはディスエーブルにして、割り込みをトリガする以下のイベントを指定できます: **On Load**、**On Store** または **On Reset**。

Clock Select（クロック選択）

あらゆる信号がシフト レジスタ コンポーネント クロック入力として使用できます。データはクロック入力信号の立ち上がりエッジでシフトされます。

配置

シフト レジスタ コンポーネントは UDB アレイリソースを使用して実装されます。必要な UDB リソースは、ツール配置アルゴリズムによって割り当てられます。

リソース

	デジタルブロック				API メモリ（バイト）		
分解能	データパス セル	PLD	ステータス セル	コントロール/カウンタ7 セル	フラッシュ	RAM	ピン（外部入出力当たり）
8 ビット	1	1	1	1	554	4	6
16 ビット	2	1	1	1	630	6	6
24 ビット	3	1	1	1	668	8	6
32 ビット	4	1	1	1	668	10	6

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。

デフォルトでは、PSoC Creator が、デザイン中のコンポーネントの最初のインスタンスにインスタンス名 “ShiftReg_1” を割り当てます。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。読みやすいように、以下の表ではインスタンス名 “ShiftReg” を使用します。

関数	説明
ShiftReg_Start()	シフト レジスタを起動し選択されたすべての割り込みをイネーブルにします
ShiftReg_Stop()	シフト レジスタをディスエーブルにします
ShiftReg_EnableInt()	シフト レジスタ割り込みをイネーブルにします
ShiftReg_DisableInt()	シフト レジスタ割り込みをディスエーブルにします
ShiftReg_SetIntMode()	割り込みに割り込みソースを設定します
ShiftReg_GetIntStatus()	シフト レジスタ割り込みステータスを取得します
ShiftReg_WriteRegValue()	シフト レジスタに直接値を書き込みます
ShiftReg_ReadRegValue()	シフト レジスタから現在の値を読み出します
ShiftReg_WriteData()	シフト レジスタ入力 FIFO にデータを書き込みます
ShiftReg_ReadData()	シフト レジスタ出力 FIFO からデータを読み出します
ShiftReg_GetFIFOStatus ()	入力または出力 FIFO の現在のステータスを返します
ShiftReg_Sleep()	コンポーネントを停止し、すべての非保持レジスタを保存します
ShiftReg_Wakeup()	すべての非保持レジスタを復元しコンポーネントを起動します
ShiftReg_Init()	デフォルトのシフト レジスタ設定を初期化または復元します
ShiftReg_Enable()	シフト レジスタをイネーブルにします
ShiftReg_SaveConfig()	シフト レジスタの設定を保存します
ShiftReg_RestoreConfig()	シフト レジスタの設定を復元します

グローバル変数

変数	説明
ShiftReg_initVar	シフト レジス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、ShiftReg_Start() が初めて呼び出されたときに 1 に設定されます。これにより、ShiftReg_Start() ルーチンを初めて呼び出した後に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。 再初期化が必要な場合は、ShiftReg_Init() 関数を ShiftReg_Start() または ShiftReg_Enable() 関数の前に呼び出すことができます。

void ShiftReg_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。ShiftReg_Start() は initVar 変数を設定し、ShiftReg_Init() 関数を呼び出し、ShiftReg_Enable() 関数を呼び出します。

Production PSoC 3 シリコンの場合、この関数を呼び出した後、コンポーネント ロジックを開始するためには 1 コンポーネント クロック パルスが必要です。

パラメータ: なし

返り値: なし

副作用: initVar 変数がすでに設定されている場合、この関数は、ShiftReg_Enable() 関数のみを呼び出します。

void ShiftReg_Stop(void)

説明: シフト レジスタをディスエーブルにします。

パラメータ: なし

返り値: なし

副作用: なし

void ShiftReg_EnableInt(void)

説明: シフト レジスタ割り込みをイネーブルにします。

パラメータ: なし

返り値: なし

副作用: なし

void ShiftReg_DisableInt(void)

説明: シフト レジスタ割り込みをディスエーブルにします。

パラメータ: なし

返り値: なし

副作用: なし

void ShiftReg_SetIntMode(uint8 interruptSource)

説明: 割り込みの割り込みソースを設定します。複数のソースは OR で結合されることがあります。

パラメータ: uint8 InterruptSource: 選択された割り込みソースのための定数を含むビット フィールド。
複数のソースを一緒に OR 演算して、複数の割り込みを選択できます。

返り値	説明
ShiftReg_LOAD_INT_EN	ロード割り込みをイネーブルにします
ShiftReg_STORE_INT_EN	ストア割り込みをイネーブルにします
ShiftReg_RESET_INT_EN	リセット割り込みをイネーブルにします

返り値: なし

副作用: なし

uint8 ShiftReg_GetIntStatus(void)

説明: シフト レジスタ割り込みの割り込みステータスを取得します

パラメータ: なし

返り値: 選択した割り込みソースのステータスを含むビット フィールド。

返り値	説明
ShiftReg_LOAD	ロード割り込みが発生しました
ShiftReg_STORE	ストア割り込みが発生しました
ShiftReg_RESET	リセット割り込みが発生しました

副作用: 割り込みステータス レジスタをクリアします。

void ShiftReg_WriteRegValue(uint8/16/32 shiftData)

説明: シフト レジスタに直接値を書き込みます。

パラメータ: uint8/16/32 shiftData: 書き込むデータ。データ タイプは Shift Register Length パラメータによって決まります。

返り値: なし

副作用: この API 関数を使用するにはコンポーネントを停止する必要があります。

注 書き込まれた値は、1 コンポーネント クロック期間後に、読み取ることができます。

uint8/16/32 ShiftReg_ReadRegValue(void)

説明: シフト レジスタから現在の値を返します。

パラメータ: なし

返り値: uint8/16/32 シフト レジスタ値。データ タイプは Length パラメータによって決まります。

副作用: シフト レジスタ出力 FIFO をクリアします。ShiftReg_WriteRegValue() を呼び出した後、この関数を呼び出す前に、最低 1 コンポーネント クロック待ちます。

注意

ShiftReg_ReadRegValue() API ルーチンと **Use Store** 出力 FIFO 機能と併用する場合は注意が必要です。ShiftReg_ReadRegValue() API 実装は、現在のシフト レジスタ ALU 値を出力 FIFO に転送し、FIFO からそのデータを読み出します。ストア信号を使用して出力 FIFO にキャプチャされたが、アプリケーションによってまだ読み出されていないデータは失われます。

cystatus ShiftReg_WriteData(uint8/16/32 shiftData)

説明: シフト レジスタ入力 FIFO にデータを書き込みます。データ ワードはロード入力の立ち上がりエッジでシフト レジスタに転送されます。

パラメータ: uint8/16/32 shiftData: 書き込むデータ。データ タイプは Shift Register Length パラメータによって決まります。

返り値: cystatus: FIFO が満杯の場合はエラーを返し、操作が正常に完了した場合は CYRET_SUCCESS を返します。入力 FIFO が満杯の場合、データは FIFO に書き込まれません。

返り値	説明
CYRET_SUCCESS	操作が正常に完了
CYRET_INVALID_STATE	入力 FIFO が満杯です

副作用: なし

uint8/16/32 ShiftReg_ReadData(void)

説明: シフト レジスタ出力 FIFO からデータを読み出します。データ ワードはストア入力の立ち上がりエッジで出力 FIFO に転送されます。

パラメータ: なし

返り値: uint8/16/32: 次に使用できるデータ ワード。データ タイプは Shift Register Length パラメータによって決まります。

副作用: なし

uint8 ShiftReg_GetFIFOStatus (uint8 fifoId)

説明: 入力または出力 FIFO の現在のステータスを返します。

パラメータ: uint8 FifoID: どの FIFO ステータスが読み取られたかを識別します。

FifoID 値	説明
ShiftReg_IN_FIFO	入力 FIFO のステータスの読み出しに使用されます
ShiftReg_OUT_FIFO	出力 FIFO のステータスの読み出しに使用されます

返り値: uint8: 定義された値のうち1 つの FIFO ステータス。

返り値	説明
ShiftReg_RET_FIFO_FULL	FIFO が満杯です
ShiftReg_RET_FIFO_NOT_FULL	FIFO が満杯ではありません
ShiftReg_RET_FIFO_EMPTY	FIFO が空です

副作用: なし

void ShiftReg_Sleep(void)

説明: これは、スリープのためにコンポーネントを準備するのに推奨されるルーチンです。ShiftReg_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、ShiftReg_Stop() 関数を呼び出し、ShiftReg_SaveConfig() 関数を呼び出してハードウェア構成を保存します。

CyPmSleep() または CyPmHibernate() 関数を呼び出す前に ShiftReg_Sleep() 関数を呼び出します。パワーマネジメント関数については、PSoC Creator *System Reference Guide (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

返り値: なし

副作用: なし

void ShiftReg_Wakeup(void)

説明: これは、ShiftReg_Sleep() が呼び出される時の状態にコンポーネントを復元するための、優先ルーチンです。ShiftReg_Wakeup() 関数は ShiftReg_RestoreConfig() 関数を呼び出して設定を復元します。ShiftReg_Sleep() 関数が呼び出される前にコンポーネントがイネーブルにされた場合、ShiftReg_Wakeup() 関数もコンポーネントを再有効化します。

Production PSoC3 シリコンの場合、この関数が呼び出された後、通常の操作に戻るには、1 コンポーネント クロック パルスが必要です。

パラメータ: なし

返り値: なし

副作用: まず ShiftReg_Sleep() または ShiftReg_SaveConfig() 関数を呼び出すことなく ShiftReg_Wakeup() 関数を呼び出すと、予期しない動作が発生することがあります。

void ShiftReg_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。ShiftReg_Start() ルーチンがこの関数を呼び出し、それがコンポーネントの動作を開始する優先的な方法であるため、ShiftReg_Init() 関数を呼び出す必要はありません。

パラメータ: なし

返り値: なし

副作用: すべてのレジスタは、Configure ダイアログに従って値が設定されます。

void ShiftReg_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。ShiftReg_Start() ルーチンがこの関数を呼び出し、この方法はコンポーネントの動作を開始する優先的な方法であるため、ShiftReg_Enable() を呼び出す必要はありません。

パラメータ: なし

返り値: なし

副作用: なし

void ShiftReg_SaveConfig(void)

説明: 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] (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、ShiftReg_Sleep() 関数に呼び出されます。

パラメータ: なし

返り値: なし

副作用: なし

void ShiftReg_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、コンポーネントのパラメータ値を ShiftReg_Sleep() 関数を呼び出す前の状態に復元します

パラメータ: なし

返り値: なし

副作用: このルーチンを呼び出す前に必ず ShiftReg_SaveConfig() 関数を呼び出します。ShiftReg_SaveConfig() 関数とは別に呼び出すと、現在の設定が初期設定で上書きされます。

定義

ShiftReg_SR_SIZE – シフト レジスタの長さをビット単位で定義します。

ShiftReg_USE_INPUT_FIFO – 入力 FIFO がプロジェクトで定義されていることを示します。

注 出力 FIFO はソフトウェア キャプチャで使用されるため、必ず定義されます。

ShiftReg_FIFOSize – シフト レジスタ ワード単位で入力 FIFO のサイズを定義します。シフト レジスタ ワード サイズは、**Length** パラメータ値によって決まります (バイト単位)。

ShiftReg_DIRECTION – シフトの方向を定義します (0 = 左シフト、1 = 右シフト)。

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。

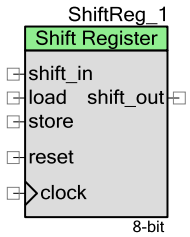
詳しくは、PSoC Creator ヘルプの「Find Example Project (サンプルプロジェクトを検索)」を参照してください。

機能の説明

シフト レジスタ パラメータにより、コンポーネントの設定の柔軟性が大幅に改善します。このセクションでは、シフト レジストリの追加説明を提供し、お使いのアプリケーション用にコンポーネントをカスタマイズするために使用するパラメータについて説明します。ソフト レジスタは、単独で使用するか、または他のコンポーネントと併用してアプリケーション固有の機能を構築できます。

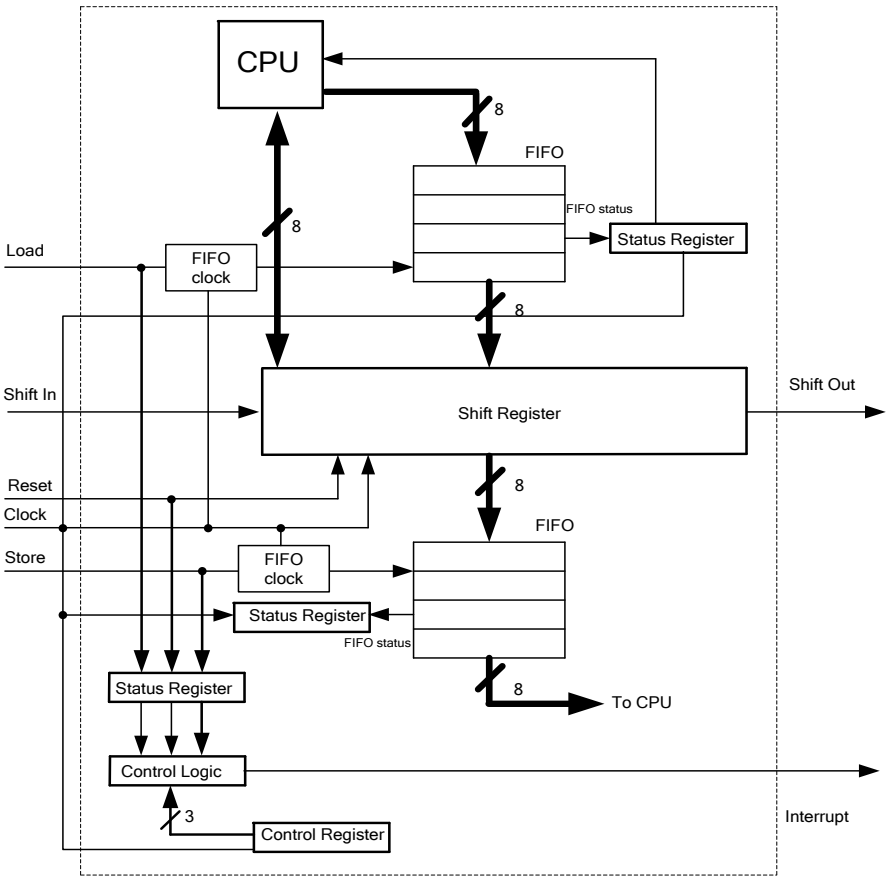
デフォルト設定

シフト レジスタ コンポーネントのデフォルト設定は、標準 7400 シリーズのロジック シフト レジスタに類似した基本的なパラレルシフト レジスタ機能を提供します。この機能には、クロック入力の立ち上がりエッジで、パラレルレジスタへの入力、またはレジスタからの出力データの同期シフトが含まれます。シリアル ビット ストリーム データは shift_in 端子へシフト入力され、shift_out 出力端子からシフト出力されます。



ブロックダイアグラムと設定

図 1. シフト レジスタ ブロック図



シフト レジスタは、入力 FIFO (F0)、ダイレクト シフト レジスタ (ソフトウェア キャプチャを提供するための重複値のある A0 および A1)、出力 FIFO (F1)、およびコントロールとステータス レジスタで構成される UDB ベースのコンポーネントです。

入力 FIFO F0 は、入力モードに設定されています。つまり、この FIFO は CPU によって書き込みでき (ShiftReg_WriteData() API 関数を使用)、この値はシフトのために A0 レジスタにロードできることを意味します。この関数は、ShiftReg_GetFIFOStatus() 関数を使用して、各サイクル前に 現在の FIFO ステータスをチェックします。

ShiftReg_WriteRegValue() を呼び出すことによって、シフトする値を A0 レジスタに直接書き込むこともできます。内部ハードウェア実装のために、ShiftReg_WriteRegValue() を使用する際には、コンポーネントの操作を停止することを強くお勧めします。(ShiftReg_Stop() 関数を使用するか、入力クロックを停止)。そうしないと、シフト操作中に A0 レジスタに書き込むことによって正しくないデータが書き込まれる場合があります。

ロード操作には、ロード イベントは入力 FIFO が空でない場合しか提供できないというハードウェア制限があります。

シフト機能を提供するには、以下の設定で UDB データパスが使用されます。

State == 100 (4)	シフト操作 (左または右)
State == 101 (5)	リセット(XOR A0 A0)
State == 110 (6)	ロード A0 <=F0
State == 111 (7)	リセット (XOR A0 A0)

ストア以外のすべての操作は、データパス コントロール ストアから制御されます。シフトがデフォルト操作です (cs_addr = “000”)。ロード入力は cs_addr[1] 配線に、リセット入力は cs_addr[0] に接続されます。これらの配線の一部は、レベルが変化すると即座に制御ストアアドレスが変わります。データパス クロックのポジティブ エッジ (この場合はコンポーネント クロック) で対応する操作が実行されます。ロードにより、ロード値が F0 から A0 に変わります。リセット コマンドにより A0 がクリアされます。この場合、ロード値は無視されます。

ハードウェア キャプチャとソフトウェア キャプチャの 2 つの仕組みを使用してシフトした値を読み出します。ハードウェア キャプチャ イベントは、ストア入力の各ポジティブ エッジで発生します。これにより、シフト レジスタ値が出力 FIFO に書き込まれます。この値は、ShiftReg_ReadData() API 関数を使用して読み出させます。ストア入力には、出力 FIFO が満杯でない場合のみストア入力が有効になるというハードウェア制限があります。

ソフトウェア キャプチャは、ShiftReg_ReadRegValue() 関数が呼び出されるたびに発生します。この関数は、A0 の値を複製した A1 値を読み出します。この操作により、A1 値は出力 FIFO F1 に自動的に書き込まれます (F1 は、A1 からのソフトウェア キャプチャ用に設定されているため)。ソフトウェア キャプチャを提供する前に、ShiftReg_ReadRegData() 関数は出力 FIFO をクリアします。このため、慎重に使用する必要があります。

注 この関数を使用することで、シフト レジスタへの書き込み後の次のクロック サイクルで A1 の実際の値が使用できるようになります。

割り込み生成メカニズムは、ステータス レジスタを使用して実装されます。これには 3 つの割り込みソース、ロード、ストア、およびリセットを表す 3 つのビットがあります。これらのビットのいずれかの値が 0 から 1 に変更されると、適切なステータス レジスタ出力の割り込みパルスが自動的に生成されます。これらの 3 つのビットは “clear-on-read” (読み出し後消去) モードです。

第 2 のステータス レジスタは、現在の入力および出力 FIFO ステータスのストアに使用されます。すべてのステータス ビットは、“sticky” (スティッキー) モードにあります (読み出し後消去されません)。

シフト レジスタ サイズが 8 を超える場合は、16、24、32 のコンポーネント サイズを実装するために 2、3、または 4 つのデータパスを相互に接続するためのデータパスのチェーン接続が提供されます。データパス幅と一致しないシフト レジスタ サイズを実装するには、データパスのコンフィギュレーションと共に verilog 制御の MSB が使用されます。

コンポーネントは、制御レジスタの CLK_EN ビットを使用して起動および停止されます。

レジスタ

ShiftReg_SR_CONTROL

ビット	7	6	5	4	3	2	1	0
値	予約済み							clk_en

- clk_en : シフト レジスタの操作をイネーブルにします

ShiftReg_SR_STATUS

ビット	7	6	5	4	3	2	1	0
値		F1_not_ empty	F1_full	F0_not_full	F0_empty	リセット	保存	ロード

- load: ステータス ビットをロードします
- store: ステータス ビットを保存します
- reset: ステータス ビットをリセットします
- F0_empty: 入力 FIFO が空です
- F0_not_full: 入力 FIFO が満杯ではありません
- F1_full: 出力 FIFO が満杯です
- F1_not_empty: 出力 FIFO は空ではありません

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。

「公称配線での最大」タイミング特性

パラメータ	説明	設定	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	–	–	66	MHz
		24 ビット	–	–	58	MHz
		32 ビット	–	–	55	MHz
t_{CLOCKH}	入力クロック高時間 [1]	該当せず	–	0.5	–	$1/f_{CLOCK}$
t_{CLOCKL}	入力クロック低時間 [1]	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 [2]	1	–	–	STA [3]	ns
t_{PD_ps}	入力パス遅延、同期ピン [4]	2	–	–	8.5	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_si}	同期出力から入力パスへのへの遅延 (配線)	1,2,3,4	–	–	STA [3]	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_{IH}	入力High時間	1,2	t_{CY_clock}	–	–	ns
t_{IL}	入力Low時間	1,2	t_{CY_clock}	–	–	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}	–	–	ns
t_{IL}	入力Low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タイムです

[2] 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。ここに記載されている数字は、多くの入力における STA 分析を基にした公称値です。

[3] t_{PD_ps} および 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周波数に直接の影響を及ぼします。静的タイミング分析結果に値がある必要があります。

[4] t_{PD_ps} 構成 2 で、デバイスのピン毎に定義された固定値。ここに記載されている数字は、デバイス上で使用できるすべてのピンの公称値です。

「すべての配線での最大」タイミング特性

パラメータ	説明	設定	Min	Typ	Max [5]	単位
f_{CLOCK}	コンポーネント クロック周波数	8 ビット	–	–	33	MHz
		16 ビット	–	–	33	MHz
		24 ビット	–	–	29	MHz
		32 ビット	–	–	27	MHz
t_{CLOCKH}	入力クロック高時間 [6]	該当せず	–	0.5	–	$1/f_{CLOCK}$
t_{clockL}	入力クロック低時間 [6]	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 [7]	1	–	–	STA [8]	ns
t_{PD_ps}	入力パス遅延、同期ピン [9]	2	–	–	8.5	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_si}	入力パス遅延への同期出力 (ルート)	1,2,3,4	–	–	STA [8]	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_{IH}	入力High時間	1,2	t_{CY_clock}	–	–	ns
t_{IL}	入力Low時間	1,2	t_{CY_clock}	–	–	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}	–	–	ns
t_{IL}	入力Low時間	1,2,3,4	t_{CY_clock}	–	–	ns

[5]すべての配線タイミング数の最大値は公称配線タイミング数を 2 分の 1 に減らして計算します。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は、このコンポーネントのタイミング合わせは問題になりません。

[6] $t_{CY_clock} = 1/f_{CLOCK}$. これは 1 クロック周期のサイクルタイムです

[7] 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。ここに記載されている数字は、多くの入力における STA 分析を基にした公称値です。

[8] t_{PD_ps} および 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周波数に直接の影響を及ぼします。静的タイミング分析結果に値がある必要があります。

[9] t_{PD_ps} 構成 2 で、デバイスのピン毎に定義された固定値。ここに記載されている数字は、デバイス上で使用できるすべてのピンの公称値です。

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。以下の方法を使用して、STA 結果を使った設計の最大値を計算できます:

f_{CLOCK} 最大コンポーネント クロック周波数は、名前付き外部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。下記の図は、*_timing.html* からのクロック限界の例を示します:

-Clock Summary

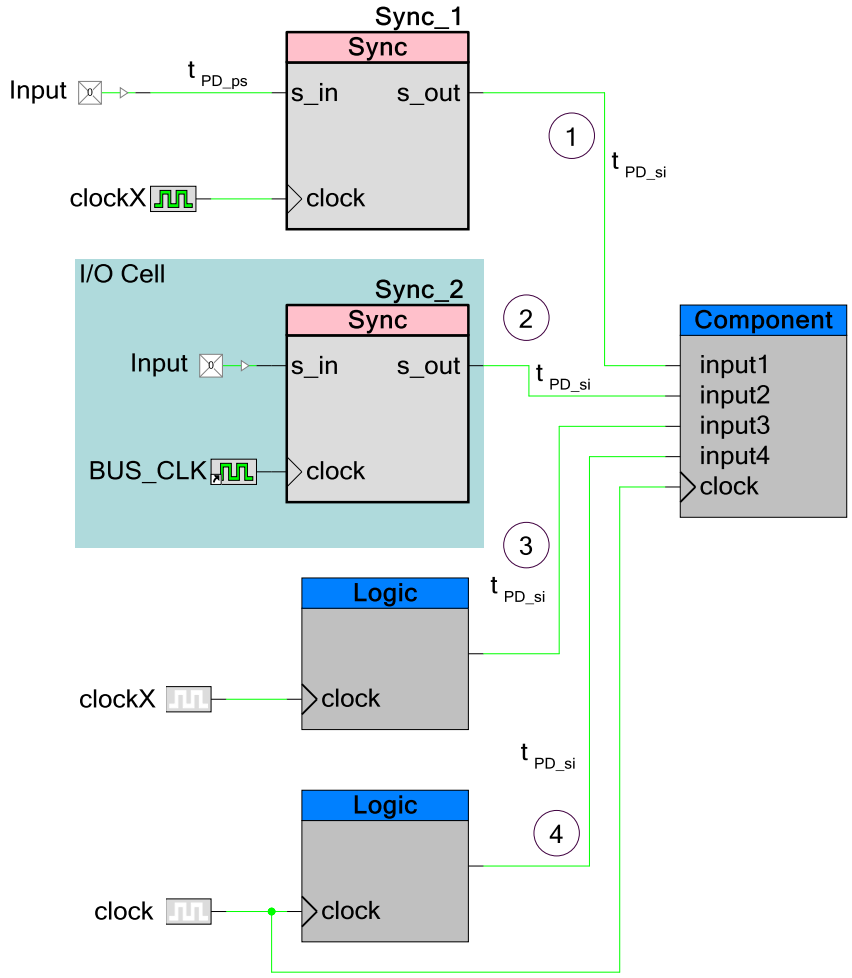
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

入力パス遅延とパルス幅

入力機能の特長を示すと、設定に関わらず、すべての入力が図 2 に示されている 、考えられる 4 つの設定のいずれかになります。

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、各入力でどの入力設定を設定したか、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 2. コンポーネントタイミング仕様のための入力設定



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7
1	クロック	master_clock	図 5
1	クロック	clockX = clock [10]	図 3
1	クロック	clockX > clock	図 4
1	クロック	clockX < clock	図 6
2	master_clock	master_clock	図7

[10] 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
2	クロック	master_clock	図 5
3	master_clock	master_clock	図 12
3	クロック	master_clock	図 10
3	クロック	clockX = clock [10]	図 8
3	クロック	clockX > clock	図 9
3	クロック	clockX < clock	図 11
4	master_clock	master_clock	図 12
4	クロック	クロック	図 8

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って同期します。このコンポーネントは、コンポーネントが使用するクロックとは異なる内部クロックを使用します (すべての内部クロックは master_clock から派生します)。

   この方法で設定される入力を特性化する場合、clockX は、コンポーネント クロックより高速、低速、または同じになります。また、図 3、図 4、図 6、図 7 に示されている特性パラメータを生成する、master_clock と同一の場合もあります。

2. この入力は、デバイスピンによって駆動され、ピンのところで、master_clock を使って同期されます。

   このようにコンフィギュレーションされ入力を評価する場合、master_clock はコンポーネントのクロックより速いか同一です。（遅いことはありません）。これにより、図 4 および図 7 の特性パラメータが生成されます。

図 3. 入力設定 1 および 2; Sync Clock Frequency = Component Clock Frequency (clock および clockX のエッジアライメントは保証されません)

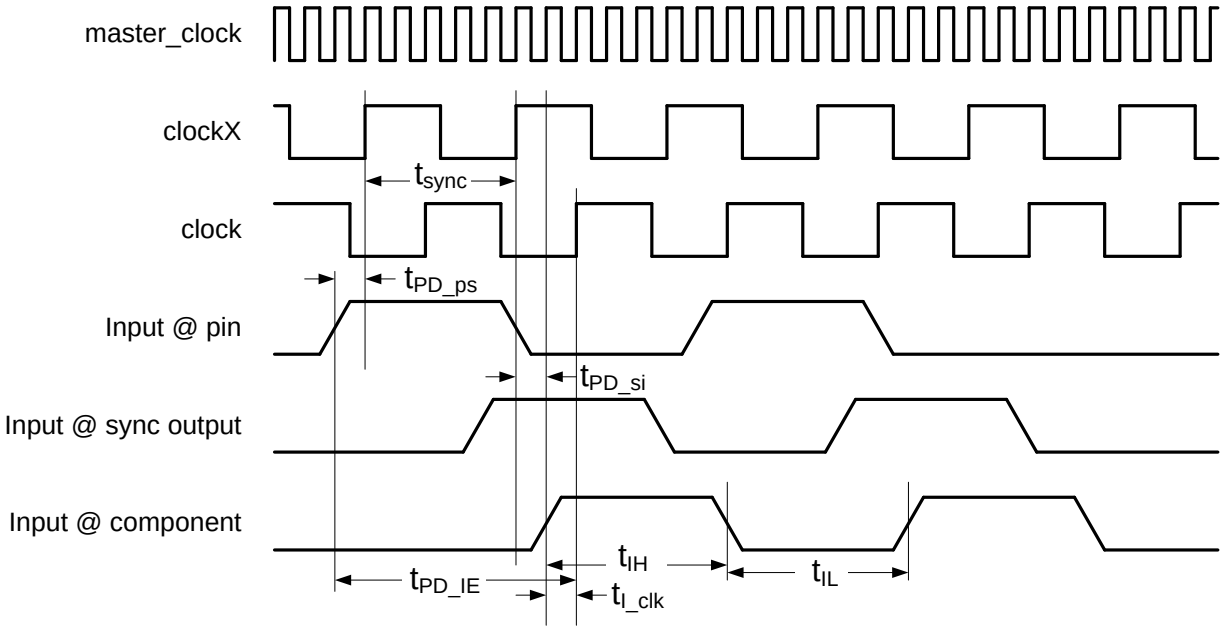


図 4. 入力設定 1 および 2; Sync Clock Frequency > Component Clock Frequency

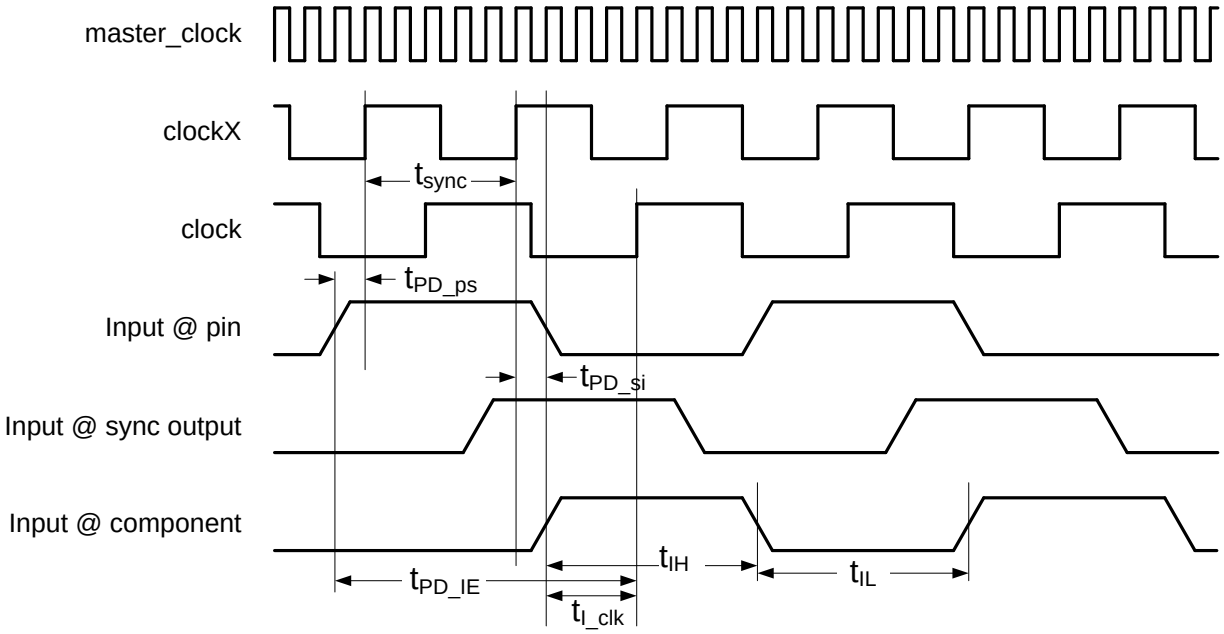


図 5. 入力設定 1 および 2;[Sync Clock Frequency == master_clock] > Component Clock Frequency

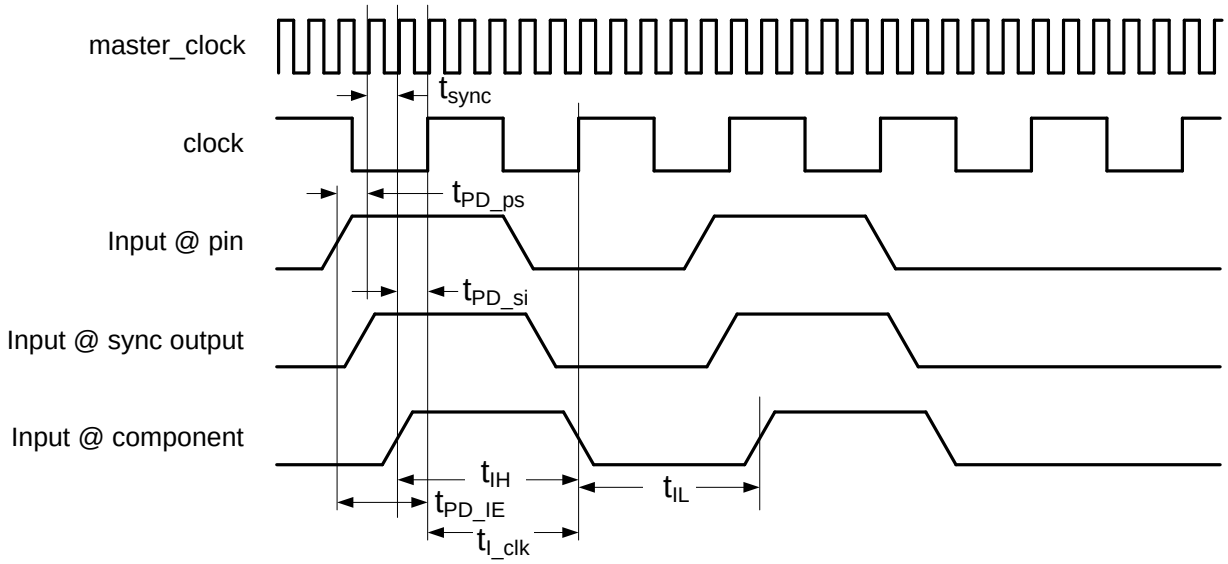


図 6. 入力設定 1; Sync Clock Frequency < Component Clock Frequency

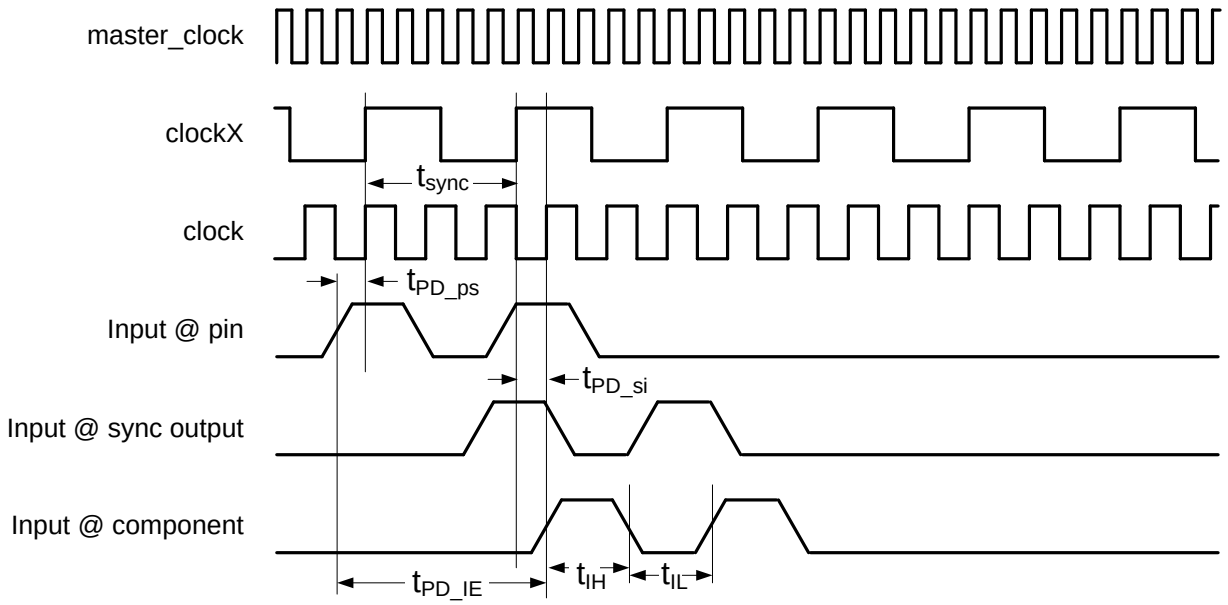
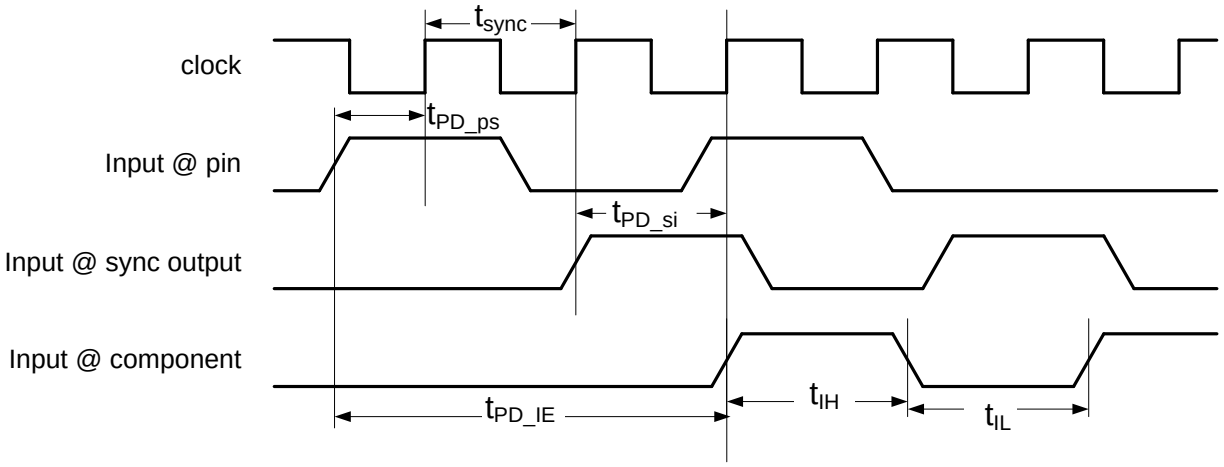


図 7. 入力設定 1 および 2; Sync Clock = Component Clock = master_clock



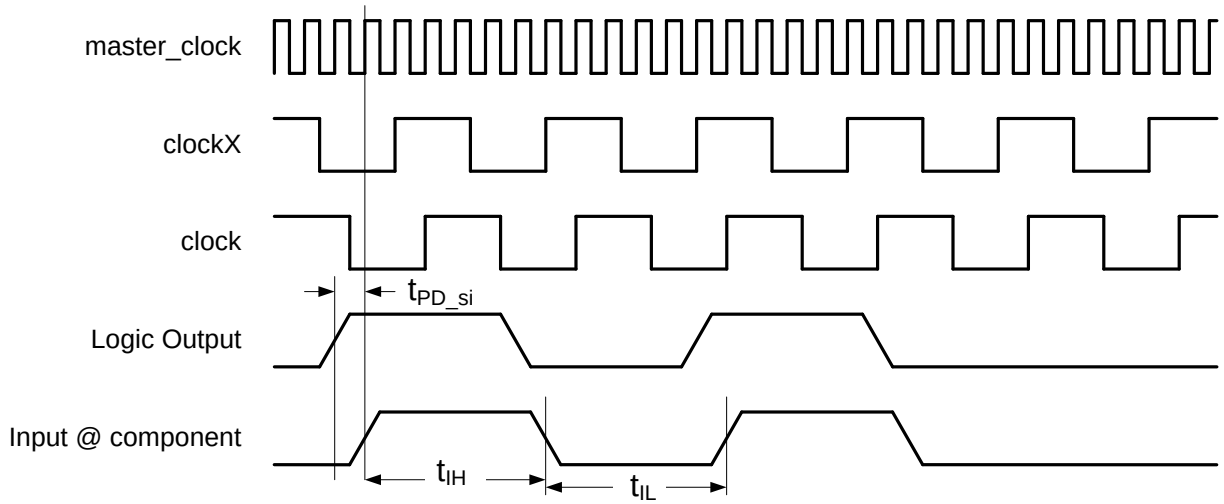
1. 入力は PSoC 内部のロジックに駆動されます。これはコンポーネントが使用するクロックとは異なるクロックをベースにして同期しています（すべての内蔵クロックは master_clock によって駆動しています）。

   特性入力をこのように設定すると、シンクロナイザ クロックは、図 8、図 9、図 11 の特性パラメータを生成するコンポーネント クロックより速い、遅い、または同じになります。

2. 入力は PSoC 内部のロジックで駆動されます。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。

   特性入力をこのように設定すると、シンクロナイザ クロックは、図 12 の特性パラメータを生成するコンポーネント クロックと同じになります。

図 8. 入力設定 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)



この数字は、静的タイミング分析がクロックに対して持つ理解を示します。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。このため、静的タイミング分析ツールは、どのクロックのエッジに同期するかが特定できず、master_clock サイクルの最低値だと推定するほかありません。これは、t_{PD_si} がシステムの master_clock に限定的な影響を与えるようになるということです。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 9. 入力設定 3; Sync Clock Frequency > Component Clock Frequency

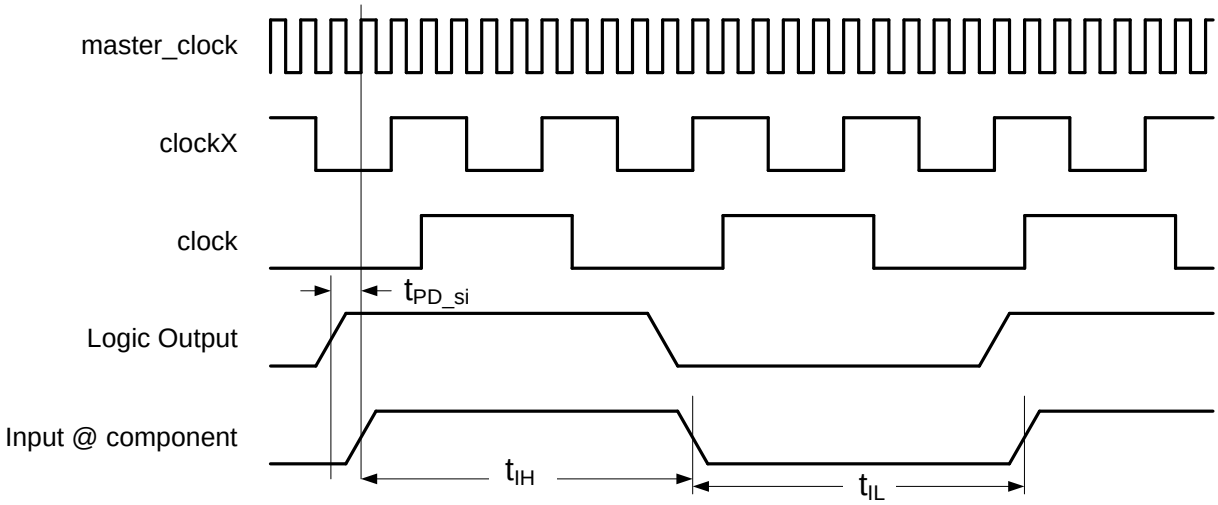


図 8 に示されているように、すべてのクロックは master_clock から派生しています。STA は、この構成で1master_clock サイクル分、master_clock における t_{PD_si} の制限を示します。このパスの遅延が長すぎると、master_clock セットアップ時間の違反が発生します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 10. 入力設定 3; Synchronizer Clock Frequency = master_clock > Component Clock Frequency

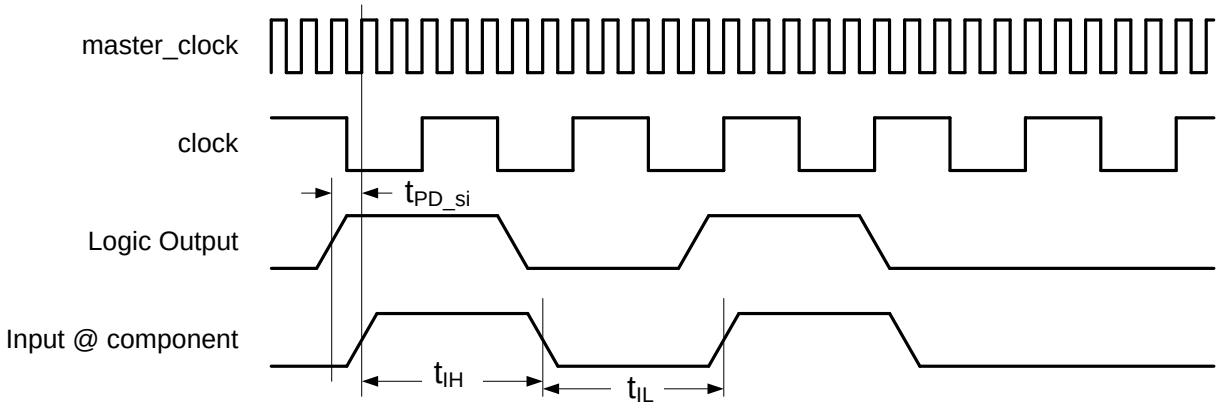


図 11. 入力設定 3; Synchronizer Clock Frequency < Component Clock Frequency

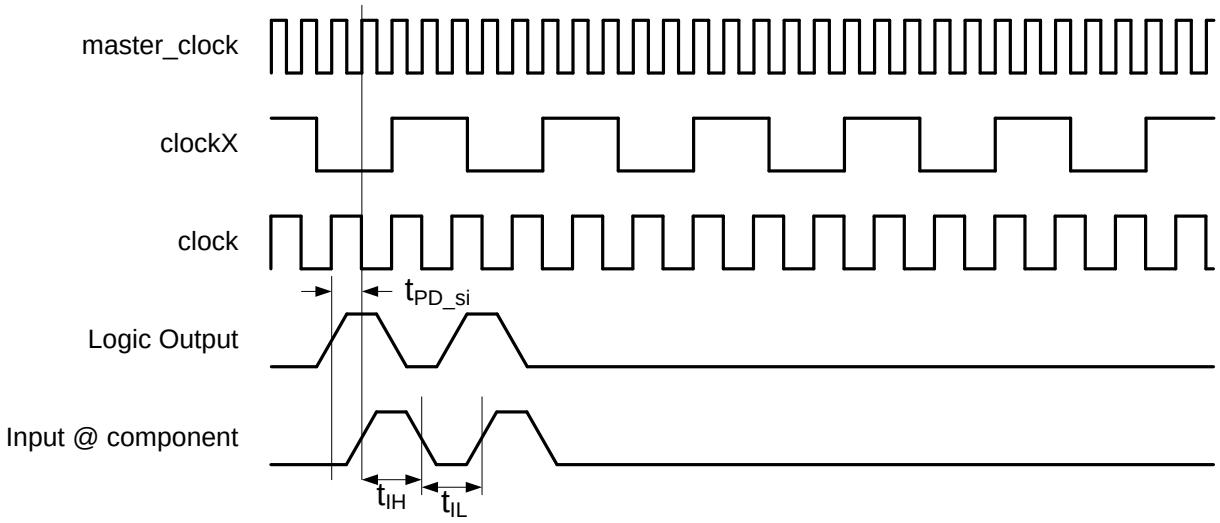
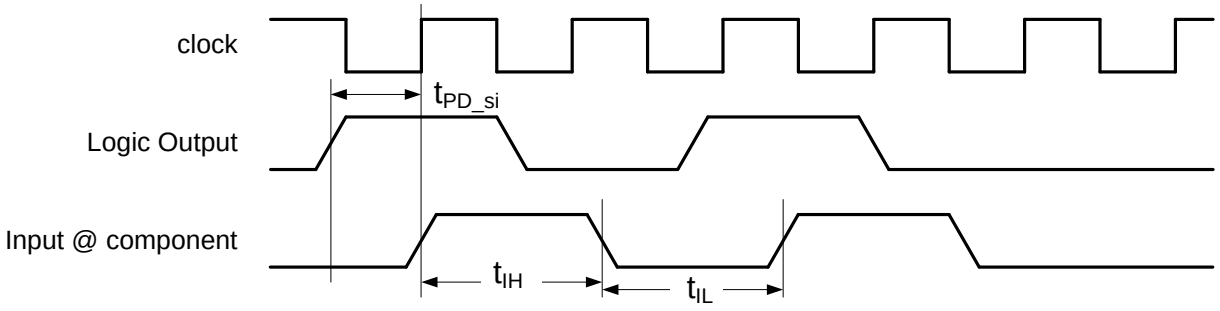


図 8 に示されているように、すべてのクロックは master_clock から派生しています。STA は、この構成で 1master_clock サイクル分、master_clock における t_{PD_si} の制限を示します。このパスの遅延が長すぎると、master_clock セットアップ時間の違反が発生します。この場合、システムの同期クロックを変更するか、master_clock を遅い周波数で実行しなければなりません。

図 12. 入力設定 4 のみ; Synchronizer Clock = Component Clock



このセクションの前のすべての図で、実装を理解する上で最も重要なパラメータは、f_{CLOCK} および t_{PD_IE} です。t_{PD_IE} は、t_{PD_ps} および t_{SYNC} (設定 1 および 2 のみ)、t_{PD_si}、および t_{I_Clk} によって定義されます。最も重要なのは、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ことです。 t_{I_Clk} は STA 結果に含まれませんが t_{PD_IE} が登録されたことを示すために使用されます。これは、シンクロナイザとコンポーネント クロックの間のルート後のマージン左です。

t_{PD_ps} と t_{PD_si} は、STA 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。この入力のファンアウトが 1 を超える場合があるため、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、レジスタ～レジスタ間伝達時間と定義されています。*_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。このパスのファンアウトは 1 以上でよく、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、デバイスの外部に出すためピンに送られるかのどちらかです。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～出力時間を参照します。

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	ロードロジックの実装を、レベル センシティブではなくエッジ センシティブに変更。	ロード信号の実装は必要に応じて変更されています。レベル センシティブ負荷が使用されるアプリケーションは、このバージョンのコンポーネントと互換性がありません。
	データシート変更: ■ リソース利用状況表を更新。 ■ ロードおよび保存信号の説明を修正。 ■ ShiftReg_Start() および ShiftReg_Wakeup() API 関数の説明を更新。	
1.60.a	データ シートの訂正	
1.60	DPクロックに対する再サンプルされたFIFOブロックステータス信号。	これで、すべての PSoC 3 および PSoC 5 シリコンの同じタイミング結果で、コンポーネントが機能します。
	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	
1.50	Sleep/Wakeup (スリープ/ウェイクアップ) と Init/Enable (初期化/イネーブル) API を追加しました。	低パワー 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イネーブル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。
	Configure ダイアログを更新。	操作性を改善するため、‘Use Shift Out’ と ‘Use Shift' の位置を変更し、‘Use interrupt’ チャックボックスのデフォルト値を変更。
	ShiftReg_ReadRegValue() 実装を変更。	これによりソフトウェア キャプチャがより速く実行できます。

バージョン	変更の説明	変更の理由 / 影響
1.20	FIFO サイズの選択のオプションは、ロードおよび保存が使用されないときにディスエーブルにされます。 [Configure] ダイアログが更新されました。 使用していないパラメータのために生成されたコードを削除しました。	完全に機能していなかったバージョン 1.10 の問題を修正するため、さまざまな点が変更されました。

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。