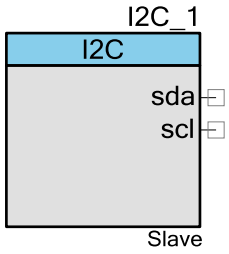
I²C マスタ/マルチマスタ/スレーブ

3.1



特長

- 業界標準 NXP® I²C バス インターフェイス
- スレーブ、マスタ、マルチマスタおよびマルチマスタ スレーブ処理をサポートします。
- 2 つのピン(SDA および SCL)だけが I²C バスとのインターフェイスに必要です。
- 100/400/1000 kbps の標準データ転送速度をサポート
- ユーザ プログラミングを最小限に抑える高レベル API

概要説明

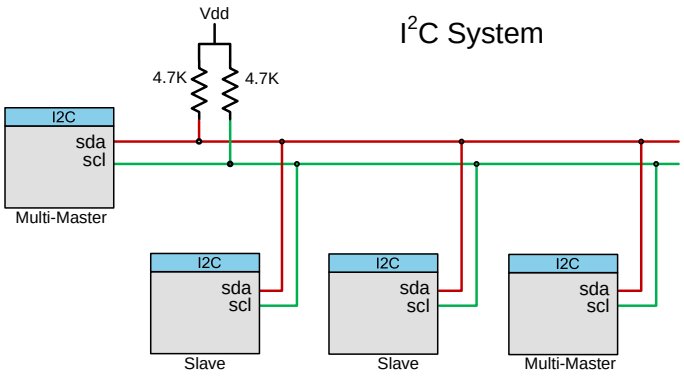
I²C コンポーネントは I²C スレーブ、マスタ、およびマルチマスタ構成をサポートします。I²C バスは、Philips 社によって開発された、業界標準の 2 線式ハードウェア インターフェイスです。マスタは、I²C バスの全通信を開始し、すべてのスレーブ デバイスへクロックを供給します。

I²C コンポーネントは最大 1000 kbps までの標準クロック速度をサポートします。I²C コンポーネントは他社のサードパーティ スレーブおよびマスタ デバイスと互換性があります。

注 コンポーネント・データシートのこのバージョンは固定ハードウェア I²C ブロックと UDB バージョンの両方に対応しています。

I²C コンポーネントの用途

I²C コンポーネントは、単一基板または小さなシステムの複数デバイスをネットワーク接続する場合、理想的なソリューションです。システムは 1 つのマスタと複数のスレーブ、複数マスタ、または複数のマスタと複数のスレーブの組み合わせで設計可能です。



入出力接続

ここでは、I²C コンポーネント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、シンボルから隠されている可能性があることを示します。

sda – 入力／出力

シリアル データ（SDA）は I²C データ信号です。これはすべてのバス データを送受信するために使用される双方向データ信号です。Sda に接続されたピンは Open-Drain-Drives-Low として設定してください。

scl – 入力／出力

シリアル クロック（SCL）はマスタで生成される I²C クロックです。スレーブは決してクロック信号を生成しませんが、データまたは ACK/NAK を送る準備ができるまで、クロックを Low レベルに保持してバスをストールさせることがあります。[1]最新のデータまたはアドレス scl に接続されたピンは Open-Drain-Drives-Low として設定してください。

clock – 入力 *

クロック入力は **Implementation** パラメータが **UDB** に設定されいる場合、使用できます。UDB バージョンはオーバーサンプリングを 16 回するためにクロックが必要です。

---

[1] NAK（否定応答）は negative acknowledgment または not acknowledged の略語です。I²C 文書では一般的に NACK を使用していますが、他のネットワーク世界では NAK を使用しています。両者は同じことを意味しています。

バス	クロック
50 kbps	800 kHz
100 kbps	1.6 MHz
400 kbps	6.4 MHz
1000 kbps	16 MHz

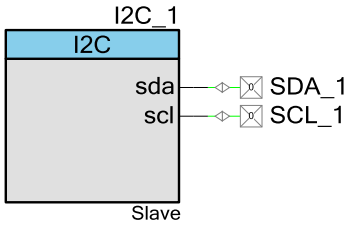
reset – 入力 *

リセット入力は **Implementation** パラメータが **UDB** に設定されいる場合、使用できます。リセット ピンが論理 HIGH に維持されている場合、I²C ブロックはリセットに維持され、I²C の通信が停止します。これはハードウェア リセットのみです。ソフトウェアは I2C_Stop() および I2C_Start() API を使用して独自にリセットする必要があります。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なることがあります。何もリセット ラインに接続していない場合、コンポーネントはそれに論理 0 に固定します。

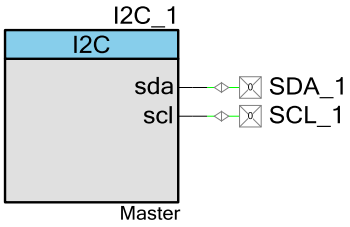
回路図マクロ情報

初期設定で、PSoC Creator Component Catalog には、I²C コンポーネントの 4 つの回路図マクロ実装が入っています。それらのマクロには、すでにピンが接続、設定された内容が含まれ、必要に応じて、クロック ソースを提供しています。回路図マクロは、下記のように I²C スレーブおよびマスタ コンポーネントを使用し、固定ファンクションおよび UDB ハードウェア用のものがあります。

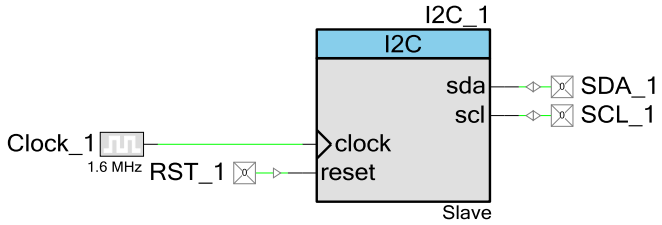
固定ファンクション I²C ピン付きスレーブ



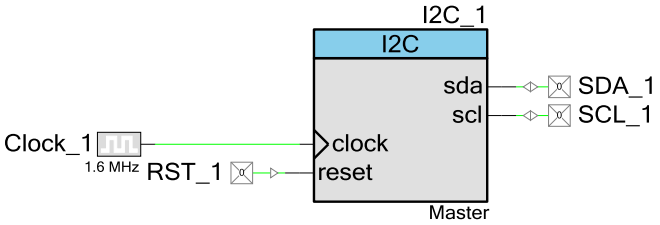
固定ファンクションI²C マスタ ピン



UDB I²C クロックおよびピン付きスレーブ

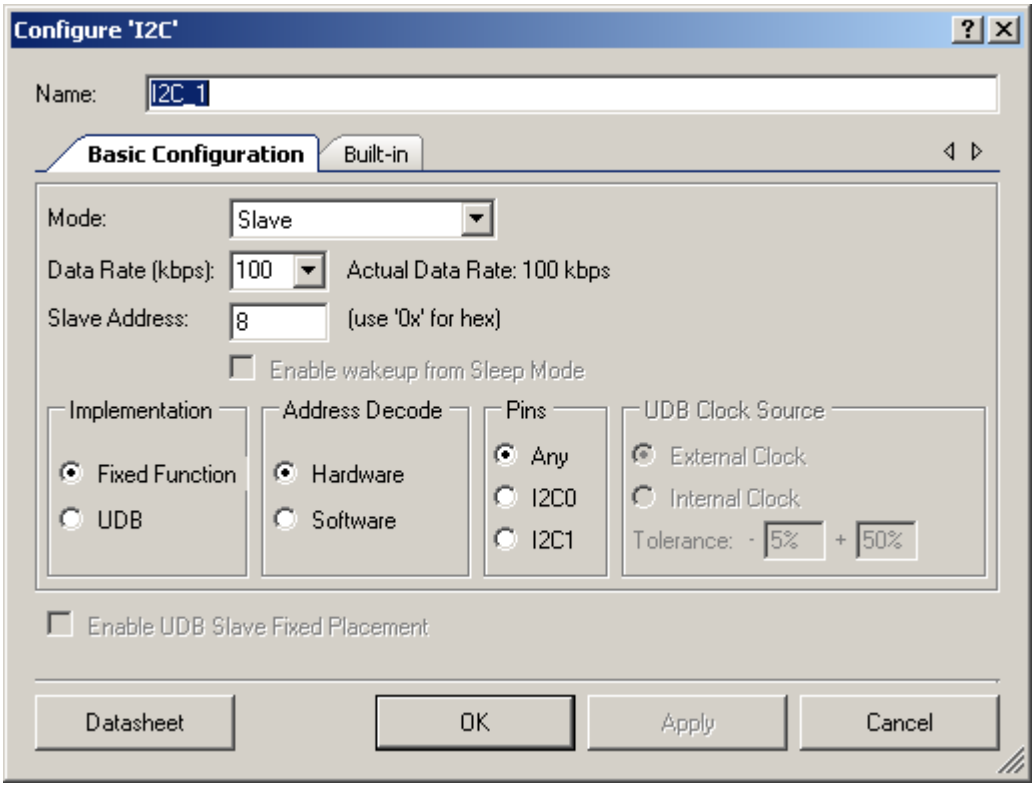


UDB I²C クロックおよびピン付きマスタ



コンポーネント・パラメータ

I²C 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。



I²C コンポーネントには次のパラメータがあります。

モード

このオプションはスレーブ、マスタ、マルチマスタ、またはマルチマスタスレーブのどのモードがサポートされるかを決定します。

モード	説明
スレーブ	スレーブのみの処理（初期設定）。
マスタ	マスタのみの処理。
マルチマスタ	バス上で複数のマスタをサポートします。
マルチマスタスレーブ	スレーブおよびマルチマスタ動作を同時使用。

データ転送速度

このパラメータは、最大 1000 kbps までの I²C データの転送速度設定に使用されます。実際の速度は使用可能なクロック速度および分周器器のレンジによる速度とは異なる場合があります。標準データ転送速度[2]は 50、100（初期設定）、400、および 1000 kbps。**Implementation** が **UDB** に設定され、**UDB Clock Source** パラメータが **External Clock** に設定されると、**Data Rate** パラメータは無視され、16x の入力クロックがデータ転送速度を決定します。

Slave Address (スレーブアドレス)

これは I²C アドレスであり、スレーブによって認識されます。スレーブ処理が選択されていない場合、このパラメータは無視されます。0 から 127（0x00 から 0x7F）の間でスレーブアドレスを選択できます。初期設定は **8** です。このアドレスは 7 ビットの右揃えスレーブアドレスで、R/W ビットを含んでいません。値を 10 進数または 16 進数として入力でき、16 進数の数字の場合、アドレスの前に「0x」と入れます。10 ビットのスレーブアドレスが必要な場合には、ソフトウェアによるアドレスのデコードを使用して、ISR 内で、10 ビットアドレスの 2 番目のバイトのデコードを行わなくてはなりません。

Implementation (実装)

このオプションは I²C ハードウェアがデバイスに実装される方法を決定します。

Implementation (実装)	説明
固定ファンクション	デバイスの固定ファンクションブロックを使用します（初期設定）。
UDB	I²C を UDB アレイに実装します。

Address Decode

このパラメータにより、ソフトウェアかハードウェアのアドレスデコードのいずれかを選択できます。提供された API が十分で、スレーブアドレスが 1 つだけあるほとんどのアプリケーションの場合、ハードウェアアドレスのデコードを推奨します。ソース コードを修正して複数のスレーブアドレスの検出を選択するアプリケーションでは、ソフトウェアアドレス検出を使用する必要があります。**Hardware** がデフォルトです。ハードウェアアドレスのデコードがイネーブルの場合、ブロックは自動的に CPU による介入無しに、自身のアドレスでないものに NAK します。正しいアドレス受信時に自動的に CPU に割り込みをかけ、CPU の介入まで SCL ラインを Low レベルにに保持します。

Pins

このパラメータは SDA および SCL 信号接続に使用するピンのタイプを決定します。以下の 3 つの値が使用可能です：**Any**、**I2C0**、および **I2C1**。初期設定は **Any** です。

[2]固定ファンクションでの実装は PSoC 3 ES2 および PSoC 5 デバイスの場合、標準データ転送速度 50、100 または 400 kbps のみをサポートします。UDB ベースの実装は最大 1000 kbps までの別のデータ転送速度の代わりに使用してください。

Any とは汎用の I/O（GPIO または SIO）を意味します。**Enable wakeup from Sleep Mode** が要求されていない場合、SDA および SCL は **Any** を使用してください。**Enable wakeup from Sleep Mode** が要求されている場合、**I2C0** または **I2C1** を使用してください。**I2C0** か **I2C1** のいずれかを使用すると、I²C アドレス一致時に起動するデバイスとして設定できます。

I²C コンポーネントは正しいピン配置であるかの確認をしません。

値	Pins
Any	回路図ルーティングを経由する Any GPIO または SIO ピン
I2C0	SCL = SIO ピン P12[4]、SDA = SIO ピン P12[5]
I2C1	SCL = SIO ピン P12[0]、SDA = SIO ピン P12[1]

Sleep Mode からのウェークアップのイネーブル化

このオプションにより、アドレス一致が発生した時、スリープ状態からシステムを復帰することができます。**Address Decode** が **Hardware** に設定され、SDA および SCL 信号が SIO ピン（**I2C0** または **I2C1**）に接続される場合、このオプションのみが有効です。オプションは初期設定でディスエーブルです。このオプションは PSoC 3 ES2 および PSoC 5 デバイスでサポートされません。

スリープモードへの切り替え中に、I²C に対しスレーブ・アドレス一致時によるウェイクアップをイネーブルしなくてはなりません。I2C_Sleep() API を呼び出すことで、これを実行することができます。詳しくは、ハードウェア・アドレス一致のウェークアップ セクションおよび『*システム・リファレンスガイド*』の「ハードウェア・アドレス一致のウェークアップ」セクションを参照してください。

UDB Clock Source（UDB クロックソース）

このパラメータにより、データ転送速度生成用に内部設定クロックか外部設定クロックのいずれかを選択できます。**Internal Clock** に設定すると、PSoC Creator は 16 回のオーバーサンプリングを勘案して **Data Rate** パラメータに基づいた必要クロック周波数を計算し、設定します。**External Clock** モードで、コンポーネントはデータ転送速度を制御できませんが、ユーザーが接続したクロックソースに基づいた実際のデータ転送速度は表示できます。このパラメータが **Internal Clock** に設定されると、クロック入力はシンボルで表示されなくなります。

所望の内部クロック公差値を入力できます。クロック公差はパーセントで指定します。スレーブモードの初期設定範囲は **-5% ～ +50%** です。クロックはこのモードでは速くできます。残りのモードの場合、初期設定範囲は -25% ～ +5% です。また、マスタは低速にすることができます。最高データ転送速度（1000 kbps）で、クロックは同じか、それより低くなることはありますが、それよりも速くなることはありません。これは予期されない動作を引き起こすことがあります。

Enable UDB Slave Fixed Placement

このパラメータにより、非拘束位置のコンポーネント性能を向上させる固定コンポーネント位置を選択できるようになります。このパラメータが設定されていると、コンポーネントリソースのすべてがデバイスの右上角に固定されます。

このパラメータはコンポーネントに接続されたピン割り当てを制御します。ピン割り当ての選択はコンポーネント性能の決定因子ではありません。**Mode** が **Slave** に設定され、**Implementation** が **UDB** に設定されている場合、このオプションのみが有効です。このオプションは初期設定でディスエーブルです。

コンポーネントを固定配置することで、「Maximum with All Routing」のケースで説明されるばらつきを解消できます（詳細は DC 電気的特性と AC 電気的特性（UDB 実装）を参照）。またこれにより、固定配置は、ほぼ空のデザイン中に非固定配置のデザインを配置するのと同様の処理を継続することができます。

Clock Select（クロック選択）

内部クロック設定が選択されている場合、PSoC Creator は必要な周波数およびクロックソースを計算し、実装用リソースを生成します。選択されていない場合、クロック コンポーネントを提供し、必要なクロック周波数を計算する必要があります。その周波数は所望のデータ転送速度 の 16x です。例えば、100 kbps のデータ転送速度では、1.6MHz のクロックが必要です。

固定ファンクションブロックは BUS_CLK を使用し、カスタマイザ分周器で計算され、16/32 のオーバーサンプリングレート（50 kbps 時のオーバーサンプリングレートは 32、他のすべてのときは 16 ）を実現しています。

注 エラッタ項目 49 シリコンの初期バージョンにおける I²C 固定ファンクションブロックで必要とする ²C クロック供給を参照してください。

リソース

リソース利用情報に使用されていた構成設定を以下に示します：(1) **Address Decode** は **Software** に設定。(2) **Enable wakeup from Sleep Mode** は選択解除。**UDB Clock Source** は **External Clock** に設定。

固定 I²C ブロックは固定ファンクションの実装に使用されています。

Mode (モード)	リソースのタイプ	API メモリ（バイト）		ピン（外部入出力ごと）
	I²C 固定ブロック	フラッシュ	RAM	
スレーブ	1	916	22	2
マスタ	1	1737	20	2
マルチマスタ	1	1889	20	2
マルチマスタスレーブ	1	2550	34	2

UDB 実装の場合、次の表を参照してください。

Mode (モード)	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	PLD	ステータス セル	Control/ Count7 セル	フラッシュ	RAM	
スレーブ	1	12	1	2	962	18	4
マスタ	2	14	1	1	1834	17	4
マルチマスタ	2	18	1	1	2007	17	4
マルチマスタ スレーブ	2	32	1	2	2754	30	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、実行中に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。

初期設定では、PSoC Creator はインスタンス名「I2C_1」を設計上のコンポーネントの最初のインスタンスに割り当てます。インスタンス名は、識別子の構文ルールに従った固有の値に変更できます。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、下表では「I2C」というインスタンス名を使用しています。

すべての API 関数では、データ方向は I²C マスタの視点から見た方向と仮定しています。データがマスタからスレーブに書き込まれると、書き込みイベントが発生します。マスタがスレーブからデータを読み取ると、読み取りイベントが発生します。

汎用関数

このセクションでは、I²C スレーブまたはマスタ処理に対し汎用な関数について説明します。

関数	説明
I2C_Start()	I²C コンポーネントを初期化してイネーブルにします。I²C 割り込みをイネーブルにして、コンポーネントが I²C トラフィックに応答できるようにします。
I2C_Stop()	I²C トラフィックへの応答を停止します(I²C 割り込みはディスエーブル)。
I2C_EnableInt()	割り込みをイネーブルにします。これはほとんどの I²C 処理に必要です。
I2C_DisableInt()	割り込みをディスエーブルにします。I2C_Stop() API はこれを自動実行します。
I2C_Sleep()	I²C 処理を停止し、I²C 非保存期間設定レジスタを保存します(割り込みはディスエーブル)。スリープ モードからのウェークアップがイネーブルの場合、アドレス一致処理の起動準備をします(I²C 割り込みはディスエーブル)。

関数	説明
I2C_Wakeup()	I²C 非保存期間設定レジスタを復元し、I²C 処理をイネーブルにします（I²C 割り込みはイネーブル）。
I2C_Init()	カスタマイザが提供した初期値を用いて I²C レジスタを初期化します。
I2C_Enable()	I²C 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。
I2C_SaveConfig()	I²C 非保存設定レジスタを保存します（I²C 割り込みをディスエーブルします）。
I2C_RestoreConfig()	I2C_SaveConfig() or I2C_Sleep() で保存された I²C 非保存設定レジスタを復元します（I²C 割り込みをイネーブルします）。

グローバル変数

それらの変数の知識は通常の処理には必要ありません。

変数	説明
I2C_initVar	I2C_initVar は I²C コンポーネント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、I2C_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、コンポーネントは I2C_Start() ルーチンへの最初の呼び出し後、再初期化なしに再起動できます。 コンポーネントの再初期化が必要な場合、I2C_Start() や I2C_Enable() 関数の前に、I2C_Init() 関数が呼び出されます。
I2C_state	I²C ステートマシンの現在の状態。
I2C_mstrStatus	I²C マスタの現在の状態。
I2C_mstrControl	Stop の生成の有無にかかわらず、マスタのトランザクションの終了を制御します。
I2C_mstrRdBufPtr	マスタ読み取りバッファへのポインタ。
I2C_mstrRdBufSize	マスタ読み取りバッファのサイズ。
I2C_mstrRdBufIndex	マスタ読み取りバッファ内の現在のインデックス。
I2C_mstrWrBufPtr	マスタ書き込みバッファへのポインタ。
I2C_mstrWrBufSize	マスタ書き込みバッファのサイズ。
I2C_mstrWrBufIndex	マスタ書き込みバッファ内の現在のインデックス。
I2C_slStatus	I²C スレーブの現在の状態。
I2C_slAddress	I²C スレーブのソフトウェアアドレス。
I2C_slRdBufPtr	スレーブ読み取りバッファへのポインタ。
I2C_slRdBufSize	スレーブ読み取りバッファのサイズ。
I2C_slRdBufIndex	スレーブ読み取りバッファ内の現在のインデックス。
I2C_slWrBufPtr	スレーブ書き込みバッファへのポインタ。

変数	説明
I2C_slWrBufSize	スレーブ書き込みバッファのサイズ。
I2C_slWrBufIndex	スレーブ書き込みバッファ内の現在のインデックス。

汎用関数

void I2C_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。I2C_Start() は I2C_Init() 関数を呼び出してから、I2C_Enable() 関数を呼び出します。I2C_Start() は I²C バス処理の前に呼び出す必要があります。

この API が I²C 割り込みをイネーブルにします。割り込みはほとんどの I²C 処理に必要です。

この関数を呼び出す前に I²C スレーブ バッファをセットアップして部分的なデータの読み取りまたは書き込みを回避する必要があります。

I²C スレーブの振る舞いは、これがイネーブルでバッファがセットアップされていない場合、次のようになります。

I²C 読み取り転送 – 読み取りバッファがセットアップされるまで 0xFF を返します。
I2C_SlaveInitReadBuf() 関数を使用して読み取りバッファをセットアップします。

I²C 書き込み転送 – 受信したデータを保管する場所がないため NAK を送ります。
I2C_SlaveInitWriteBuf() 関数を使用して書き込みバッファをセットアップします。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_Stop(void)

説明: この関数は I²C ハードウェアおよび割り込みをディスエーブルにします。

FF 実装(**PSoC 3 製品版限定**): デバイスによってバスがロックアップされている場合、I²Cバスを開放し、アイドル状態します。

UDB の実装 デバイスによってバスがロックアップされている場合、I²Cバスを開放し、アイドル状態します。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_EnableInt(void)

説明: この関数は I²C 割り込みをイネーブルにします。割り込みはほとんどの処理に必要です。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_DisableInt(void)

説明: この関数は I²C 割り込みをディスエーブルにします。この関数は I2C_Stop() 関数が割り込みをディスエーブルにしているため、通常は不要です。

パラメータ: なし

返り値: なし

副作用: I²C がなおも実行し続けているときに I²C 割り込みがディスエーブルされると、I²C バスのロックアップが発生する可能性があります。

void I2C_Sleep(void)

説明: これは、コンポーネントのスリープを準備するのに推奨される API です。I²C 割り込みは関数呼び出し後、ディスエーブルになります。

アドレス一致によるウェイクアップをイネーブル: このデバイス向けのトランザクションは API 呼び出し時に実行する場合、現在のトランザクションが完了するまで待機します。このデバイス向けの後続のすべての I²C トラフィックは、デバイスがスリープに入るまで NAK 化されます。アドレス一致イベントがチップをウェークアップします。

アドレス一致のウェークアップのディスエーブル:この API が現在の I²C コンポーネント状態を確認し、それを保存して、アドレス一致のウェークアップがイネーブルの場合は、I2C_Stop() を呼び出すことでコンポーネントをディスエーブルにします。次に、I2C_SaveConfig() を呼び出して I²C 非保存設定レジスタを保存します。

CyPmSleep() および CyPmHibernate() 関数を呼び出す前に、I2C_Sleep() 関数を呼び出してください、。電源管理関数については、『*PSoC Creator 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 I2C_Wakeup(void)

説明: これは、I2C_Sleep() が呼び出されたときの状態にコンポーネントを復元するのに推奨される API です。I²C 割り込みは関数呼び出し後にイネーブルになります。

アドレス一致ウェークアップをイネーブル: この API は、スリープ前にマスタファンクションがイネーブルの場合、I²C マスタ機能をイネーブルにして、I²C バックアップ レギュレータをディスエーブルにします。I²C 割り込みをイネーブルにするとすぐ着信トランザクションが継続します。

アドレス一致ウェークアップのディスエーブル: この API は I²C 非保存設定レジスタを I2C_RestoreConfig() の呼び出しにより復元します。このコンポーネントが I2C_Sleep() 関数を呼び出す前にイネーブルにされた場合、I2C_Wakeup() 関数はコンポーネントを再度イネーブルにします。

パラメータ: なし

返り値: なし

副作用: 最初に I2C_Sleep() または I2C_SaveConfig() 関数を呼び出すことなく I2C_Wakeup() 関数を呼び出すと、予期されない振る舞いにつながる可能性があります。

void I2C_Init(void)

説明: この関数はカスタマイザの [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。I2C_Start() API が I2C_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

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 I2C_Enable(void)

説明: この関数は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。I2C_Start() API が I2C_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 API を呼び出す場合、I2C_Start() または I2C_Init()を先に呼び出す必要があります。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_SaveConfig(void)

説明: この関数は I²C コンポーネントの非保存設定レジスタを保存し、I²C 割り込みをディスエーブルにします

アドレス一致ウェークアップをイネーブル: この API は、I²C マスタが以前にイネーブルにされていた場合、これをディスエーブルにして、I²C バックアップ レギュレータをイネーブルにします。このデバイス向けのトランザクションは API 呼び出し時に実行する場合、現在のトランザクションが完了するまで待機し、I²C がスリープになる準備をします。後続のすべての I²C トラフィックは、デバイスがスリープに入るまで NAK 化されます。

アドレス一致のウェークアップのディスエーブル:メイン記述を参照してください。

I²C 割り込みをディスエーブルすることは、アドレス一致のウェークアップのイネーブル、ディスエーブルに左右されることはありません。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_RestoreConfig(void)

説明: この関数は I²C コンポーネントの非保存設定レジスタを I2C_Sleep() または I2C_SaveConfig() が呼び出される前に置かれていた状態に復元します。I²C 割り込みをイネーブルにします。

アドレス一致のウェークアップを下記のようにイネーブルにします: この API は、以前にイネーブルにしたことがあれば、I²C マスタ機能をイネーブルにして、I²C バックアップ・レギュレータをディスエーブルにします。

アドレス一致のウェークアップのディスエーブル: メイン記述を参照してください。

I²C 割り込みをイネーブルにすることは、アドレス一致のウェークアップのイネーブル、ディスエーブルに左右されることはありません。

パラメータ: なし

返り値: なし

副作用: 最初に I2C_Sleep() または I2C_SaveConfig() 関数を呼び出すことなく、この関数を呼び出すと、予期されない振る舞いにつながる可能性があります。

スレーブ関数

このセクションは I²C スレーブ処理に使用される関数を示します。これらの関数はスレーブ処理がイネーブルの場合、使用できます。

関数	説明
I2C_SlaveStatus()	スレーブステータスフラグを返します。
I2C_SlaveClearReadStatus()	読み取りステータスフラグを返し、スレーブ読み取りステータスフラグをクリアします。

I2C_SlaveClearWriteStatus()	書き込みステータスフラグを返し、スレーブ書き込みステータスフラグをクリアします。
I2C_SlaveSetAddress()	スレーブアドレス、0 から 127（0x00 ～ 0x7F）の値を設定します。
I2C_SlaveInitReadBuf()	スレーブ受信データバッファをセットアップします。（マスタ <- スレーブ）
I2C_SlaveInitWriteBuf()	スレーブ書き込みバッファをセットアップします。（マスタ -> スレーブ）
I2C_SlaveGetReadBufSize()	バッファがリセットされたため、マスタが読み込んだバイト数を返します。
I2C_SlaveGetWriteBufSize()	バッファがリセットされたため、マスタが書き込んだバイト数を返します。
I2C_SlaveClearReadBuf()	読み取りバッファのカウンタをゼロにリセットします。
I2C_SlaveClearWriteBuf()	書き込みバッファのカウンタをゼロにリセットします。

uint8 I2C_SlaveStatus(void)

説明: この関数はスレーブの通信状態を返します。

パラメータ: なし

返り値: uint8: I²C スレーブの現在の状態。

スレーブ状態定数	説明
I2C_SSTAT_RD_CMPLT[3]	スレーブの読み取り転送が完了しました。マスタが NAK を送信する時に、読み取り完了を伝えるように設定します。
I2C_SSTAT_RD_BUSY	スレーブの読み取り転送処理中。マスタがリード時にスレーブをアドレスしたときにセットされ、RD_CMPLT が設定されている場合はクリアされます。
I2C_SSTAT_RD_ERR_OVFL	マスタがバッファにあるバイトを超えて読み取ろうとしました。
I2C_SSTAT_WR_CMPLT[4]	スレーブの書き込み転送が完了しました。Stop 条件を受信した時に設定されます。
I2C_SSTAT_WR_BUSY	スレーブの書き込み転送処理中。マスタが書き込みでスレーブ対応時に設定され、WR_CMPLT が設定されている場合はクリアされます。
I2C_SSTAT_WR_ERR_OVFL	マスタがバッファの終了を過ぎて書き込もうとしました。着信バイトはスレーブで NAK 化されます。

副作用: なし

[3] - 定義がマスタ読み取り完了定義に準拠するよう I2C_SSTAT_RD_CMPT から I2C_SSTAT_RD_CMPLT に変更されました。コンポーネントは両方の定義をサポートしますが、I2C_SSTAT_RD_CMPT は廃止されます。

[4] - 定義がマスタ読み取り完了定義に準拠するよう I2C_SSTAT_WR_CMPT から I2C_SSTAT_WR_CMPLT に変更されました。コンポーネントは両方の定義をサポートしますが、I2C_SSTAT_WR_CMPT は廃止されます。

uint8 I2C_SlaveClearReadStatus(void)

説明: この関数はリードステータスフラグをクリアして、それらの値を返します。他のステータスフラグには影響しません。

パラメータ: なし

返り値: uint8: スレーブの現在のリードステータス。定数については I2C_SlaveStatus() を参照してください。

副作用: なし

uint8 I2C_SlaveClearWriteStatus(void)

説明: この関数は書き込みステータスフラグをクリアして、それらの値を返します。他の状態フラグには影響しません。

パラメータ: なし

返り値: uint8: スレーブの現在の書き込みステータス。定数については I2C_SlaveStatus() を参照してください。

副作用: なし

void I2C_SlaveSetAddress(uint8 address)

説明: この関数は I²C スレーブアドレスを設定します

パラメータ: uint8 address: プライマリデバイスの I²C スレーブアドレス。この値は 0 から 127 (0x00 ～ 0x7F)の任意の値にすることができます。このアドレスは 7 ビットの右揃えスレーブアドレスで、R/W ビットを含んでいません。

返り値: なし

副作用: なし

void I2C_SlaveInitReadBuf(uint8 * rdBuf, uint8 bufSize)

説明: この関数は読み取りバッファのバッファポインタおよびサイズを設定します。この関数は I2C_SlaveGetReadBufSize() 関数で返されたトランスファーカウントもリセットします。

パラメータ: uint8* rdBuf: マスタによって読み取られるデータバッファへのポインタ。

uint8 bufSize: I²C マスタへ公開されるバッファのサイズ。

返り値: なし

副作用: この関数がバス トランザクション中に呼び出された場合、過去のバッファ位置および現在のバッファ開始位置からのデータが送信されることがあります。

void I2C_SlaveInitWriteBuf(uint8 * wrBuf, uint8 bufSize)

説明: この関数は書き込みバッファのバッファポインタおよびサイズを設定します。この関数は I2C_SlaveGetWriteBufSize() 関数で返されたトランスファーカウントもリセットします。

パラメータ: uint8* wrBuf: マスタによって書き込まれるデータバッファへのポインタ。

uint8 bufSize: I²C マスタへ公開される書き込みバッファのサイズ。

返り値: なし

副作用: この関数がバス トランザクション中に呼び出された場合、過去のバッファ位置および現在のバッファ位置でデータが受信されることがあります。

uint8 I2C_SlaveGetReadBufSize(void)

説明: この関数は、I2C_SlaveInitReadBuf() または I2C_SlaveClearReadBuf() 関数が実行されたため、I²C マスタで読み込まれたバイト数を返します。

最大返り値は読み取りバッファのサイズです。

パラメータ: なし

返り値: uint8: マスタで読み取られたバイト。

副作用: なし

uint8 I2C_SlaveGetWriteBufSize(void)

説明: この関数は、I2C_SlaveInitWriteBuf() または I2C_SlaveClearWriteBuf() 関数が実行され、I²C マスタで書き込まれたバイト数を返します。

最大返り値は書き込みバッファのサイズです。

パラメータ: なし

返り値: uint8: マスタで書き込まれたバイト。

副作用: なし

void I2C_SlaveClearReadBuf(void)

説明: この関数はリードポインタをリードバッファの最初のバイトにリセットします。マスタが読み取る次のバイトはリードバッファの最初のバイトになります。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_SlaveClearWriteBuf(void)

説明: この関数はライトポインタをライトバッファの最初のバイトにリセットします。マスタが書き込む次のバイトはライトバッファのの最初のバイトになります。

パラメータ: なし

返り値: なし

副作用: なし

マスタおよびマルチマスタ関数

これらの関数は、マスタまたはマルチマスタモードがイネーブルの場合のみ使用可能です。

関数	説明
I2C_MasterStatus()	マスタステータスを返します。
I2C_MasterClearStatus()	マスタステータスを返し、ステータスフラグをクリアします。
I2C_MasterWriteBuf()	参照データバッファを指定のスレーブアドレスに書き込みます。
I2C_MasterReadBuf()	データを指定のスレーブアドレスから読み取り、参照バッファにデータを配置します。
I2C_MasterSendStart()	指定のアドレスに Start のみを送信します。
I2C_MasterSendRestart()	指定のアドレスに Restart のみを送信します。
I2C_MasterSendStop()	Stop 条件を生成します。
I2C_MasterWriteByte()	1 バイトを書き込みます。これは I2C_MasterSendStart() または I2C_MasterSendRestart() 関数のみに使用するマニュアル・コマンドです。
I2C_MasterReadByte()	1 バイトを読み取ります。これは I2C_MasterSendStart() または I2C_MasterSendRestart() 関数のみに使用するマニュアル・コマンドです。
I2C_MasterGetReadBufSize()	I2C_MasterClearReadBuf() 関数が呼び出されたため、データ読み取りのバイト数を返します。
I2C_MasterGetWriteBufSize()	I2C_MasterClearWriteBuf() 関数が呼び出されたとき、データ書き込みのバイト数を返します。
I2C_MasterClearReadBuf()	リードバッファポインタをバッファの開始に戻ってリセットします。
I2C_MasterClearWriteBuf()	ライトバッファポインタをバッファの開始に戻ってリセットします。

uint8 I2C_MasterStatus(void)

説明: この関数はマスタの通信ステータスを返します。

パラメータ: なし

返り値: uint8: I²C マスタの現在の状態。I²C マスタの状態定数はORされることがあります。

マスタのステータス定数	説明
I2C_MSTAT_RD_CMPLT	読み取り転送完了。 読み取り転送が成功した確認するため、エラーコンディションビットをチェックしなくてはなりません。
I2C_MSTAT_WR_CMPLT	書き込み転送完了。 書き込み転送が成功したことを確認するため、エラー条件ビットをチェックしなくてはなりません。
I2C_MSTAT_XFER_INP	転送処理中
I2C_MSTAT_XFER_HALT	転送が一時停止しました。I²C バスはマスタが Restart または Stop 条件を生成するのを待っています。
I2C_MSTAT_ERR_SHORT_XFER	エラー条件:すべてのバイトが転送される前に、ライト転送が終了しました。
I2C_MSTAT_ERR_ADDR_NAK	エラー条件:スレーブは指定したアドレスに応答しませんでした。
I2C_MSTAT_ERR_ARB_LOST	エラー条件:スレーブで通信中のマスタのアービトレーション・ロスト。
I2C_MSTAT_ERR_XFER	エラー条件:これは、この表で提供されるエラーコンディションをORした値です。 すべてのエラー条件ビットがクリアされていても、このビットがセットされているとき、転送はスレーブ動作のために中断されます。

副作用: なし

uint8 I2C_MasterClearStatus(void)

説明: この関数はすべてのステータスフラグをクリアし、マスタステータスを返します。

パラメータ: なし

返り値: uint8: マスタの現在のステータス。定数についてはI2C_MasterStatus() 関数を参照してください。

副作用: なし

uint8 I2C_MasterWriteBuf(uint8 slaveAddress, uint8 * wrData, uint8 cnt, uint8 mode)

説明: この関数は自動的にデータのバッファ全体をスレーブ デバイスに書き込みます。データ転送がこの関数で初期化された後、内蔵ISRがバイトごとのモードで以降のデータ転送を管理します。I²C 割り込みをイネーブルにします。

パラメータ: uint8 slaveAddress: 右揃え 7 ビット スレーブアドレス（有効範囲 0 ～ 127）

uint8 wrData: 送信するデータのバッファへのポインタ

uint8 cnt: 送信するバッファのバイト数

uint8 mode: 転送モードは次のようなものを定義しています:(1) 転送の開始時に Start と Restartコンディションのどちらが生成さえるか(2)バス上にStopコンディションが生成される前に転送が完了するか停止するか。

転送モード、モード定数は互いにORされることがあります。

モード定数	説明
I2C_MODE_COMPLETE_XFER	全転送を Start から Stop まで実行します。
I2C_MODE_REPEAT_START	Start の代わりに Repeat Start を送信します。
I2C_MODE_NO_STOP	Stop 無しで転送を実行します。

返り値: uint8: エラー状態。定数については、I2C_MasterSendStart() 関数を参照にしてください。

副作用: なし

uint8 I2C_MasterReadBuf(uint8 slaveAddress, uint8 * rdData, uint8 cnt, uint8 mode)

説明: この関数は自動的にスレーブ デバイスからデータバッファ全体を読み取ります。この関数がデータ転送を開始すると、内蔵ISRがバイトごとのモードで以降のデータ転送を管理します。I²C 割り込みをイネーブルにします。

パラメータ: uint8 slaveAddress: 右揃え 7 ビット スレーブアドレス（有効範囲 0 ～ 127）

uint8 rdData: スレーブからデータを配置するバッファへのポインタ

uint8 cnt: 読み取るバッファのバイト数

uint8 mode: 転送モードは次のように定義しています: (1) 転送の開始時に Start とRestartコンディションのいずれを生成するか。(2) バス上にStop コンディションが生成される前に転送が完了するか停止するか。

転送モード、モード定数は互いにORされることがあります。

モード定数	説明
I2C_MODE_COMPLETE_XFER	全転送を Start から Stop まで実行します。
I2C_MODE_REPEAT_START	Start の代わりに Repeat Start を送信します。
I2C_MODE_NO_STOP	Stop 無しで転送を実行します。

返り値: uint8: エラー状態。定数については、I2C_MasterSendStart() 関数を参照にしてください。

副作用: なし

uint8 I2C_MasterSendStart(uint8 slaveAddress, uint8 R_nW)

説明: この関数は Start 条件を生成し、読み/書きビットと共にスレーブアドレスを送信します。I²C 割り込みをディスエーブルにします。

パラメータ: uint8 slaveAddress: 右揃え 7 ビット スレーブアドレス（有効範囲 0 ～ 127）

uint8 R_nW: ゼロにセットするとライトコマンド送信、ゼロ以外の値にセットすると、リードコマンド送信。

返り値: uint8: エラー状態。

モード定数	説明
I2C_MSTR_NO_ERROR	関数はエラー無く完了しました。
I2C_MSTR_BUS_BUSY	バスビジー状態発生。スタートコンディション生成は開始されませんでした。
I2C_MSTR_NOT_READY	マスタはバス上で有効なマスタではないか、スレーブ処理が進行中です。
I2C_MSTR_ERR_LB_NAK	最後のバイトが NAK 化されました。
I2C_MSTR_ERR_ARB_LOST	Startが生成されているときにマスタがアービトレーションで負けた。（このステータスはマルチマスタが使用可能な場合にのみ有効です。）
I2C_MSTR_ABORT_XFER	Startコンディションの生成はスレーブ処理が開始されたため中断しました。（このステータスはマルチマスタスレーブ モードでのみ有効です。）

副作用: この関数はブロッキングされており、byte_complete ビットが I2C_CSR レジスタに設定されるまで終了しません。

uint8 I2C_MasterSendRestart(uint8 slaveAddress, uint8 R_nW)

説明: この関数はリスタート条件を生成し、リード／ライトビットと共にスレーブアドレスを送信します。

パラメータ: uint8 slaveAddress: 右揃え 7 ビット スレーブアドレス（有効範囲 0 ～ 127）

uint8 R_nW: ゼロにセットすると、ライトコマンド送信、ゼロ以外の値をセットするとリードコマンド送信。

返り値: uint8: エラー状態。定数については、I2C_MasterSendStart() 関数を参照にしてください。

副作用: この関数はブロッキングされており、byte_complete ビットが I2C_CSR レジスタに設定されるまで、終了しません。

uint8 I2C_MasterSendStop(void)

説明: この関数はバス上に I²C ストップコンディションを生成します。この関数は呼び出される前に Start または Restart コンディション生成が失敗した場合、何もしません。

パラメータ: なし

返り値: uint8: エラー状態。定数については I2C_MasterSendStart() を参照してください。

副作用: この関数はブロッキングされており、以下になるまで、終了しません。

Master: この関数はストップコンディション生成中は待機します。

Multi-Master, Multi-Master-Slave: この関数はストップ条件が生成されている間、または ACK/NAKビット上でアービトレーションをロストしているときに待機します。

uint8 I2C_MasterWriteByte(uint8 theByte)

説明: この関数は 1 バイトをスレーブに送信します。この関数を呼び出す前に有効な Start または Restart コンディションが生成される必要があります。この関数は呼び出される前に Start または Restartコンディション生成が失敗した場合、何もしません。

パラメータ: uint8 theByte: スレーブに送るデータ バイト。

返り値: uint8: エラー状態。

モード定数	説明
I2C_MSTR_NO_ERROR	関数はエラー無く完了しました。
I2C_MSTR_NOT_READY	マスタはバス上で有効なマスタではないか、スレーブ処理が進行中です。
I2C_MSTR_ERR_LB_NAK	最後のバイトが NAK 化されました。
I2C_MSTR_ERR_ARB_LOST	Start コンディション生成中にマスタのアービトレーションでロスト（負け）しました。（このステータスはマルチマスタがイネーブルされているときのみ有効です。）

副作用: この関数はブロッキングされており、byte_complete ビットが I2C_CSR レジスタに設定されるまで、終了しません。

uint8 I2C_MasterReadByte(uint8 acknNak)

説明: この関数はスレーブ から 1 バイトを読み取るか、転送を ACK または NAK します。この関数を呼び出す前に有効な Start または Restart コンディションを生成しなくてはなりません。この関数は呼び出される前に Start または Restartコンディション生成が失敗した場合、何もせずにゼロ値を返します。

パラメータ: uint8 acknNak: ゼロの場合、NAK を送信し、非ゼロの場合、ACK を送信します。

返り値: uint8: バイトはスレーブから読み取ります

副作用: この関数はブロッキングされており、byte_complete ビットが I2C_CSR レジスタに設定されるまで、終了しません。

uint8 I2C_MasterGetReadBufSize(void)

説明: この関数は I2C_MasterReadBuf() 関数で転送されたバイト数を返します。

パラメータ: なし

返り値: uint8: 転送のバイト数。転送がまだ完了していない場合、この関数はこれまで転送したバイト数を返します。

副作用: なし

uint8 I2C_MasterGetWriteBufSize(void)

説明: この関数 I2C_MasterWriteBuf() 関数で転送されたバイト数を返します。

パラメータ: なし

返り値: uint8: 転送のバイト数。転送がまだ完了していない場合、この関数はこれまで転送したバイト数を返します。

副作用: なし

void I2C_MasterClearReadBufSize(void)

説明: この関数は読み取りバッファポインタをバッファの最初のバイトに戻ってリセットします。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_MasterClearWriteBufSize(void)

説明: この関数は書き込みバッファポインタをバッファの最初のバイトに戻ってリセットします。

パラメータ: なし

返り値: なし

副作用: なし

マルチマスタスレーブ関数

マルチマスタスレーブはスレーブおよびマルチマスタ関数が組み込まれています。

ブートローダ サポート

I²C コンポーネントはブートローダの通信コンポーネントとして使用できます。次の設定を使用して、外部システムからブートローダへの通信プロトコルをサポートします。

- **Mode(モード)**:スレーブ
- **Implementation(実装)**:固定ファンクションまたは UDB ベースのいずれか
- **Data Rate(データ転送速度)**:ホスト(ブート デバイス)のデータ転送速度と一致する必要があります。
- **Slave Address(スレーブアドレス)**:ホスト(ブート デバイス)の選択されたスレーブアドレスと一致する必要があります。
- **Address Match(アドレス一致)**: ハードウェアは推薦しますが、必須ではありません。

ブートローダの詳細については、『*システム リファレンス ガイド*』の「ブートローダ システム」セクションを参照してください。

I²C 通信コンポーネント実装の詳細については、「I2C 通信コンポーネントによるブートローダ・プロトコルインタラクション」セクションを参照してください。

I²C コンポーネントはブートローダ用 API 関数を提供します。

関数	説明
I2C_CyBtldrCommStart	I²C コンポーネントを開始し、その割り込みをイネーブルにします。
I2C_CyBtldrCommStop	I²C コンポーネントをディスエーブルにし、その割り込みをディスエーブルにします。
I2C_CyBtldrCommReset	読み取りおよび書き込み用I²C バッファを初期状態に設定し、スレーブ状態をリセットします。
I2C_CyBtldrCommWrite	呼び出し元がデータをブートローダ ホストに書き込むことができるようになります。この関数はポーリングを管理して、データのブロックをホスト デバイスへ完全に送信できるようにします。
I2C_CyBtldrCommRead	呼び出し元がデータをブートローダ ホストに読み取ることができるようになります。この関数はポーリングを管理して、データのブロックをホスト デバイスへ完全に受信できるようにします。

void I2C_CyBtldrCommStart(void)

説明: この関数は I²C コンポーネントを開始し、その割り込みをイネーブルにします。
到達した各 I²C 書き込みトランザクションはブートローダのコマンドとして扱われます。
到達した各 I²C 読み取りトランザクションはブートローダが実行済みコマンドに対応して供給されるまで 0xFF を返します。

パラメータ: なし

返り値: なし

副作用: なし

void I2C_CyBtldrCommStop(void)

説明: この関数は、I²C コンポーネントをディスエーブルにし、その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 I2C_CyBtldrCommReset(void)

説明: この関数は読み取り書き込み I²C バッファを初期状態に設定し、スレーブ状態をリセットします。

パラメータ: なし

返り値: なし

副作用: なし

cystatus I2C_CyBtldrCommRead(uint8 * Data, uint16 size, uint16 * count, uint8 timeOut)

説明: この関数により、呼び出し元がデータをブートローダ・ホストから読み取ることができるようになります。この関数はポーリングを管理して、データのブロックをホスト・デバイスから完全に受信できるようにします。

パラメータ: uint8 *Data: デバイスに送信するデータのブロックへのポインタ
uint16 size: 書き込むバイト数
uint16 *count: 実際に書き込んだバイト数を書き込む変数へのポインタ
uint8 timeOut: タイムアウトによって復帰するまでの間の待ち時間の10ms単位の数値。

返り値: cystatus: 何も問題がなかった場合、CYRET_SUCCESS を返すか、または問題を最も適切に表現する値を返します。詳細については、『*システム・リファレンスガイド*』の「リターンコード」を参照してください。

副作用: なし

cystatus I2C_CyBtldrCommWrite(uint8 * Data, uint16 size, uint16 * count, uint8 timeOut)

説明:	この関数により、呼び出し元がデータをブートローダ・ホストに書き込むことができるようになります。この関数はポーリングを管理して、データのブロックをホスト・デバイスに完全に送信できるようにします。
パラメータ:	uint8 *Data: デバイスに送信するデータのブロックへのポインタ uint16 size: 書き込むバイト数 uint16 *count: 実際に書き込んだバイト数を書き込む変数へのポインタ uint8 timeOut: タイムアウトによってリターンするまでの待ち時間の10ms単位の数値
返り値:	cystatus: 何も問題がなかった場合、CYRET_SUCCESS を返すか、または問題を最も適切に表現する値を返します。詳細については、『*システム・リファレンスガイド*』の「リターンコード」を参照してください。
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

このコンポーネントは I²C スレーブ、マスタ、マルチマスタ、およびマルチマスタスレーブ設定をサポートします。、次のセクションでは、スレーブ、マスタ、およびマルチマスタ コンポーネントの使用方法の概要を説明します。

このコンポーネントは、I²C ハードウェアが割り込みドライブのため、グローバル割り込みをイネーブルにする必要があります。このコンポーネントでは割り込みが必要になりますが、ISR (割り込みサービスルーチン) にコードを追加する必要はありません。コンポーネント・サービスは、ユーザが記述したコードとは独立してすべての割り込みサービス (データ転送) を担います。このインタフェースに割り当てられたメモリ バッファは、アプリケーションと I²C マスタ/スレーブ間のシンプルなデュアル メモリのように見えます。

スレーブ動作

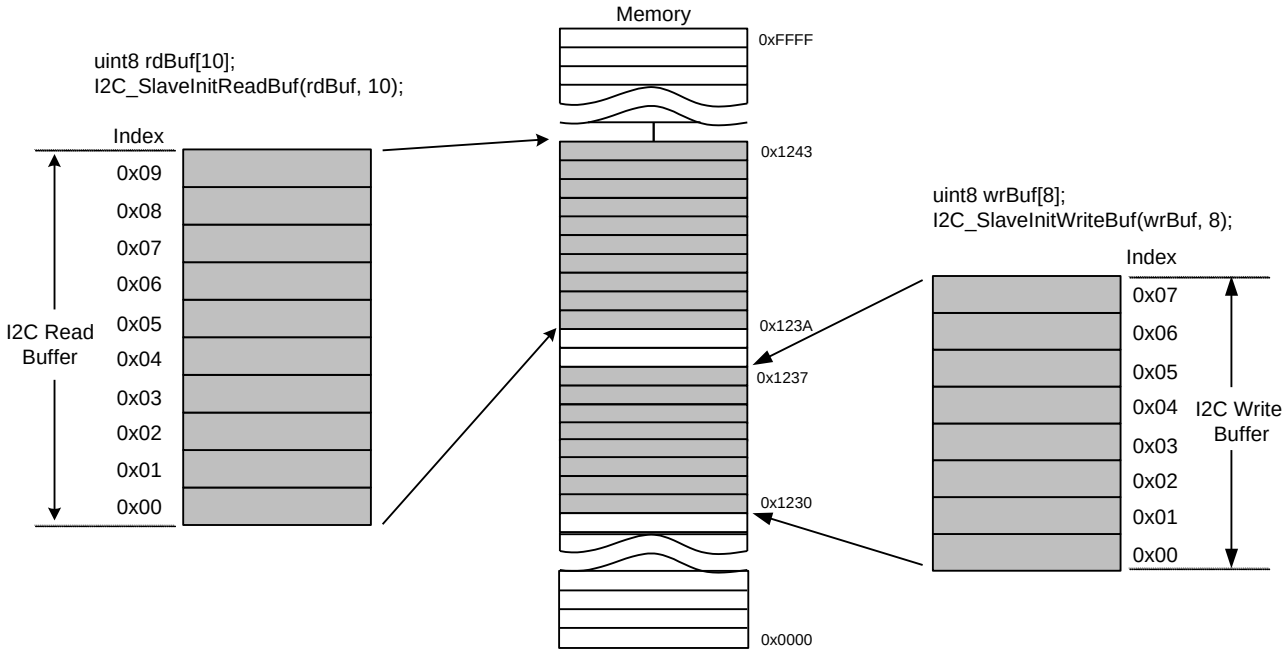
スレーブのインタフェースはメモリの 2 つのバッファから成り、一 つはマスタによるスレーブへのデータ書き込み用で、もう一つのバッファはマスタによるスレーブからのデータ読み取り用です。読み取り、書き込みは I²C マスタの視点からのため注意が必要です。I²C スレーブの読み取り及び書き込みバッファは、以下の初期化コマンドで設定されます。これらのコマンドはメモリの割り当てはありませんが、その代わりアレイ ポインタとサイズを内部コンポーネン

ト変数にコピーします。配列はコンポーネントでは自動的に生成されないため、バッファに使用するにはインスタンス化する必要があります。読み取りおよび書き込みバッファ共に同じバッファを使用できますが、注意してデータを適切に管理する必要があります。

```
void I2C_SlaveInitReadBuf(uint8 * rdBuf, uint8 bufSize)
void I2C_SlaveInitWriteBuf(uint8 * wrBuf, uint8 bufSize)
```

上記の関数を使用すると、読み取りおよび書き込みバッファのポインタおよびバイト カウントを設定できます。それらの関数の bufSize は実際の配列のサイズ以下になることがありますが、rdBuf または wrBuf ポインタでポイントされる使用可能メモリを超えることは決してありません。

図 1.スレーブ バッファ構造



I2C_SlaveInitReadBuf() または I2C_SlaveInitWriteBuf() 関数が呼び出された場合、内部インデックスは rdBuf および wrBuf のそれぞれが指し示す配列の最初の値にセットされます。I²C マスタはバイトを読み取りまたは書き込むので、インデックスはオフセットが byteCount より小さくなるまでインクリメントされます。随時、転送されたバイト数は、I2C_SlaveGetReadBufSize() または I2C_SlaveGetWriteBufSize() のいずれかを呼び出すことで読み取りおよび書き込みバッファを個々に照会できます。バッファにあるバイトを超えて読み取りまたは書き込むと、オーバーフロー エラーの原因となります。エラーはスレーブ状態のバイトに設定され、I2C_SlaveStatus() API で読み取ることができます。

アレイの最初に戻ってインデックスをリセットするには、次のコマンドを使用してください。

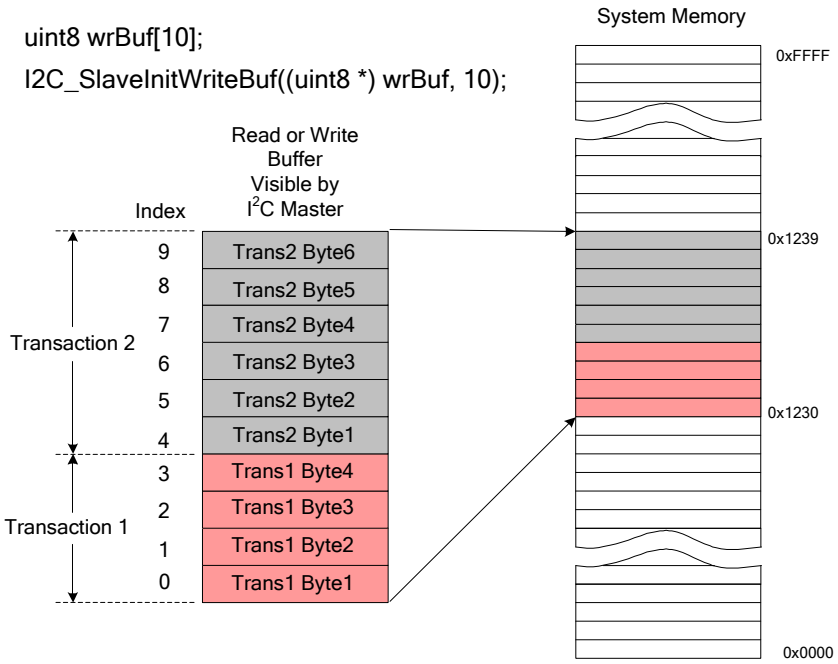
```
void I2C_SlaveClearReadBuf(void)
void I2C_SlaveClearWriteBuf(void)
```

これはインデックスをゼロに戻ってリセットします。I2C マスタが読み取るまたは書き込む次のバイトは、配列の最初のバイトになります。これらのクリア バッファ コマンドを使用する前に、アレイのデータは読み取るか更新しておいてください。

I²C マスタによる複数読み取りまたは書き込みは、クリア バッファ コマンドが使用されるか、アレイ インデックスがアレイ サイズを越えて増加を試みるまで、アレイ インデックスをインクリメントし続けます。図 2 は I²C マスタが 2 つの書き込みトランザクションを実行した例を示します。最初の書き込みは 4 バイトで、2 番目の書き込みは 6 バイトでした。2 番目のトランザクションの 6 バイト目はスレーブがバッファの最後への到達発生を示すため、NAK 応答します。2 番目のトランザクションの 7 バイトを書き込もうとしたマスタ、または 3 番目のトランザクションでさらにバイトを書き込み開始したマスタは、バッファがリセットされるまで各バイトが NAK 応答され、廃棄されます。

最初のトランザクションの後、I2C_SlaveClearWriteBuf() 関数を使用すると、ゼロに戻ってインデックスをリセットし、2 番目のトランザクションが最初のトランザクションのデータを上書きする原因となります。データはバッファをオーバフローすることで消失しないことを確認してください。バッファのデータはバッファ インデックスをリッセットする前にスレーブで処理してください。

図 2.システム メモリ



読み取りおよび書き込みバッファの両方に信号転送完了、転送処理中、およびバッファのオーバーフローを示す3つのステータスビットがあります。転送が始まるとビジーフラグが設定されます。転送が完了すると、転送完了フラグが設定されて、ビジーフラグはクリアされます。2 番目の転送が始まった場合、ビジーおよび転送完了フラグの両方が同時に設定可能です。次の表は読み取りおよび書き込み状態フラグを示しています。

スレーブ状態定数	値	説明
I2C_SSTAT_RD_CMPLT	0x01	スレーブの読み取り転送が完了しました

スレーブ状態定数	値	説明
I2C_SSTAT_RD_BUSY	0x02	スレーブの読み取り転送処理中（ビジー）
I2C_SSTAT_RD_OVFL	0x04	マスタがバッファにあるバイトを超えて読み取ろうとしました
I2C_SSTAT_WR_CMPLT	0x10	スレーブの書き込み転送が完了しました
I2C_SSTAT_WR_BUSY	0x20	スレーブの書き込み転送処理中（ビジー）
I2C_SSTAT_WR_OVFL	0x40	マスタがバッファの終了を過ぎて書き込もうとしました

次のコードの例では書き込みバッファを初期化して、転送が完了するのを待ちます。転送が完了後、データはワーク用の配列にコピーされます。多くのアプリケーションでは、データは別の場所にコピーする必要はありませんが、その代わりに元のバッファで処理することができます。ほぼ同じように読み取りバッファ例を書き込み関数および定数を読み取り関数と定数を置き換えて作成できます。新規データを表すデータを処理すると、外部の代わりにスレーブ バッファ内部に転送されます。

```
uint8 wrBuf[10];
uint8 userArray[10];
uint8 byteCnt;

/* Initialize write buffer before call I2C_Start */
I2C_SlaveInitWriteBuf((uint8 *) wrBuf, 10);

/* Start I2C Slave operation */
I2C_Start();

/* Wait for I2C master to complete a write */

for(;;)  /* loop forever  */
{
   /* Wait for I2C master to complete a write */
   if(0u != (I2C_SlaveStatus() & I2C_SSTAT_WR_CMPLT))
   {
      byteCnt = I2C_SlaveGetWriteBufSize();
      I2C_SlaveClearWriteStatus();
      for(i=0; i < byteCnt; i++)
      {
         userArray[i] = wrBuf[i]; /* Transfer data */
      }
      I2C_SlaveClearWriteBuf();
   }
}
```

マスタ/マルチマスタ処理

マスタおよびマルチマスタ[5,6]処理は基本的に同じですが、例外が 2 つあります。マルチマスタ・モードで処理する場合、プログラムは常にビジー状態かどうか確認するためバスをチェックする必要があります。他のマスタはすでに別のスレーブと通信していることがあります。この場合、プログラムは Start トランザクションを発する前に現在の処理が完了するまで待機する必要があります。プログラムは返り値を見ます。別のマスタがバスをコントロールしている場合、エラーがセットされています。

マルチマスタ・モードで、2 つのマスタが同時に開始できることが 2 つ目の違いです。もしこれが発生したら、2 つのマスタのうち 1 つがアービトレーション（バス使用権）を失います 各バイトが転送後、この状態かどうかを確認する必要があります。コンポーネントは自動的にこの状態を確認し、アービトレーションがロスとしている場合、エラーで応答します。

I²C マスタを処理する場合、次の 2 つのオプションがあります：マニュアルおよび自動。自動モードでは、バッファが作成され、転送全体が保留になります。書き込み処理の場合、バッファには送信するデータが事前に入力されています。データがスレーブから読み取られる場合、バッファは最低でもパケットサイズ分を割り当てる必要があります。バイト列を自動的にスレーブに書き込むには、次の関数を使用してください。

```
uint8 I2C_MasterWriteBuf(uint8 slaveAddress, uint8 * xferData, uint8 cnt, uint8
mode)
```

slaveAddress 変数は 0 から 127 の 7 ビットの右揃えスレーブ・アドレスです。コンポーネント API は自動的にアドレスの LSb に書き込みフラグを加えます。2 番目のパラメータ、xferData は転送するデータのアレイをポイントします。cnt パラメータは転送するバイト数です。最後のパラメータ、mode は転送を開始および停止する方法を決定します。トランザクションは Start の代わりに Restart で始めるか、または Stop シーケンスの前に一時停止することができます。これらのオプションにより、最後の転送が Stop を送信せずに次の転送が Start の代わりに Restart を発するバックツーバックの転送が可能になります。

読み取り処理は書き込み処理とほぼ同じです。同じ定数で同じパラメータを使用します。

```
uint8 I2C_MasterReadBuf(uint8 slaveAddress, uint8 * xferData, uint8 cnt, uint8
mode);
```

両方の関数ともステータスを返します。I2C_MasterStatus() 関数の返り値についてはステータス表を参照してください。読み取り及び書き込み転送は I²C 割り込みコード時にバックグラウンドで完了しているため、I2C_MasterStatus() 関数を使用していつ転送が完了したか定することができます。代表的なスレーブへの書き込みを示すコード例は以下のようになります。

---

[5]マスタまたはマルチマスタの PSoC 3 ES2 と PSoC 5 の固定ファンクションの実装で、ソフトウェアが Start コンディションの直後に Stop コンディションを設定すると、モジュールは Stop コンディションを生成します。これはアドレス フィールドの後に発生し（データ書き込みの場合、0xFF を送信）、クロック ラインは Low のままになります。この状態を避けるために、 Start 後、すぐに Stop 条件を設定しないでください。少なくとも 1 バイトを転送し、NAK または ACK の後に Stop コンディションを設定してください。

[6]固定ファンクションの実装は未定義のバス条件はサポートしません。それらの条件を避けるか、またはその代わりに UDB ベースの実装を使用してください。

```
I2C_MasterClearStatus(); /* Clear any previous status */
I2C_MasterWriteBuf(0x08, (uint8 *) wrData, 10, I2C_MODE_COMPLETE_XFER);
for(;;)
{
   if(0u != (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT))
   {

      /* Transfer complete. Check Master status to make sure that transfer
         completed without errors. */

      break;
   }
}
```

I²C マスタはマニュアルで動作させることも可能です。このモードでは、書き込みトランザクションの各パートが個々のコマンドで実行されます。

```
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);
if(status == I2C_MSTR_NO_ERROR)   /* Check if transfer completed without errors */
{
   /* Send array of 5 bytes */
   for(i=0; i<5; i++)
   {
      status = I2C_MasterWriteByte(userArray[i]);
      if(status != I2C_MSTR_NO_ERROR)
      {
         break;
      }
   }
}
I2C_MasterSendStop();      /* Send Stop */
```

マニュアル読み取りトランザクションは NAK 化する必要のある最終バイトを除き、書き込みトランザクションとほぼ同等です。以下の例は代表的なマニュアル読み取りトランザクションを示します。

```
status = I2C_MasterSendStart(0x08, I2C_READ_XFER_MODE);
if(status == I2C_MSTR_NO_ERROR)   /* Check if transfer completed without errors */
{
   /* Read array of 5 bytes */
   for(i=0; i<5; i++)
   {
      if(i < 4)
      {
         userArray[i] = I2C_MasterReadByte(I2C_ACK_DATA);
      }
      else
      {
         userArray[i] = I2C_MasterReadByte(I2C_NAK_DATA);
      }
   }
}
I2C_MasterSendStop();      /* Send Stop */
```

マルチマスタスレーブ・モードの動作

マルチマスタおよびスレーブの両方がこのモードで稼働します。コンポーネントはスレーブとしてアドレス設定できますが、ファームウェアもマスタ・モード転送を起動できます。このモードで、マスタがアドレス バイト中にアービトレーションを失うと、ハードウェアはスレーブ・モードに戻り、受信したバイトがスレーブ・アドレス割り込みを生成します。

マスタおよびスレーブ動作の例については、スレーブ動作 および マスタ/マルチマスタ処理 セクションを参照してください。

ハードウェア・アドレス一致がイネーブルでアドレス バイト制限をアービトレージします：マスタがアドレス バイト中にアービトレーションを失うと、スレーブはアドレス設定されている場合に限り、スレーブ・アドレス割り込みを生成します。その他の場合、アービトレーションロストステータスは割り込みベースの関数によって消失します。ソフトウェアのアドレス検出はこの可能性を排除しますが、ハードウェアアドレス一致機能によるウェイクアップは除外されます。

マニュアル関数の I2C_MasterSendStart() は先ほど説明したように正常な状態情報を提供します。

マルチマスタスレーブ転送の開始

マルチマスタスレーブを使用すると、スレーブはいつでもアドレス設定できます。マルチマスタはバスがフリーな場合、Start コンディションを生成する準備に時間をかける必要があります。この時間の間に、スレーブはアドレス設定できます。その場合、マルチマスタ トランザクションは失われ、スレーブ動作が進行します。スレーブ動作を妨害しないように注意してください。I²C 割り込みはトランザクションがアドレス・ステージを通過しないように Start コンディション生成前にディスエーブルにする必要があります。この動作により、マルチマスタ トランザクションを中断して、スレーブ動作を正しく開始することができます。次の事例は I²C 割り込みがディセーブルされている場合に使用可能です：

- バスは Start コンディション生成前にビジー（スレーブ動作が進行中か、他のトラフィックがバス上にある）である。マルチマスタは Start 条件の生成を試みません。スレーブ動作は I²C 割り込みがイネーブルの場合、続行します。I2C_MasterWriteBuf()、I2C_MasterReadBuf()、または I2C_MasterSendStart() 呼び出しは状態 **I2C_MSTR_BUS_BUSY** を返します。
- バスは Start 生成前はフリーです。マルチマスタは I²C 割り込みがイネーブルの場合、Start コンディションをバス上に生成し、動作を再開します。I2C_MasterWriteBuf()、I2C_MasterReadBuf()、または I2C_MasterSendStart() 呼び出しは状態 **I2C_MSTR_NO_ERROR** を返します。
- バスは Start 生成前はフリーです。マルチマスタは Start の生成を試みますが、別のマルチマスタがスレーブをアドレス設定し、バスがビジーになります。Start コンディション生成はキュー行列に入ります。スレーブ動作は I²C 割り込みがディセーブルされたことにより、アドレス・ステージで停止します。I²C 割り込みがイネーブルの場合、マルチマスタ トランザクションはキュー行列から中断され、スレーブ動作が続行されます。I2C_MasterWriteBuf() または I2C_MasterReadBuf() 呼び出しはこれを通知せず、**I2C_MSTR_NO_ERROR** を返します。The I2C_MasterStatus() はマルチマスタ トランザクション中断後に **I2C_MSTAT_WR_CMPLT** または **I2C_MSTAT_RD_CMPLT** を **I2C_MSTAT_ERR_XFER** を用いて（他のすべてのエラー条件ビットはクリアされます）返します。I2C_MasterSendStart() 呼び出しはエラー状態 **I2C_MSTR_ABORT_XFER** を返します。

割り込み関数の動作

- I2C_MasterWriteBuf();
- I2C_MasterReadBuf();

```
I2C_MasterClearStatus();     /* 前のすべてのステータスをクリアします */
I2C_DisableInt();            /* 割り込みをディスエーブルにします */

status = I2C_MasterWriteBuf(0x08, (uint8 *) wrData, 10, I2C_MODE_COMPLETE_XFER);
/* Try to generate, start. The disabled I2C interrupt halt the transaction on
address stage in case of Slave addressed or Master generates start condition */

I2C_EnableInt();             /* Enable interrupt and proceed Master or Slave
transaction */

for(;;)
{
   if(0u != (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT))
   {
     /* Transfer complete. Check Master status to make sure that transfer
        completed without errors. */
      break;
   }
}

if (0u != (I2C_MasterStatus() & I2C_MSTAT_ERR_XFER))
{
      /* Error occurred while transfer, clean up Master status and
         retry the transfer */
}
```

マニュアル関数の動作

マニュアル・マルチマスタの動作は I²C 割り込みのディスエーブルが前提ですが、次の対策をとる方が良いと思われます:

```
I2C_DisableInt();         /* Disable interrupt */
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);;    /* Try to generate
start condition */
if (status == I2C_MSTR_NO_ERROR)   /* Check if start generation completed without
errors */
{
   /* Proceed the write operation */
   /* Send array of 5 bytes */
   for (i=0; i<5; i++)
   {
      status = I2C_MasterWriteByte(userArray[i]);
      if (status != I2C_MSTR_NO_ERROR)
      {
         break;
      }
   }
    I2C_MasterSendStop();      /* Send Stop */
```

```
}
I2C_EnableInt();          /* Enable interrupt, if it was enabled before */
```

ハードウェア・アドレス一致のウェークアップ

次の条件を満足した場合、I²C・アドレス一致イベントによるスリープからのウェークアップが可能です:

- I²C スレーブはイネーブルです。スレーブまたはマルチマスタスレーブ・モードが選択されている。
- I²C ハードウェア・アドレス検出が選択されている。
- SIO ペアが SCL および SDA に接続されており、以下のように適切なペアがカスタマイザで選択されている: I2C0 – SCL P12[4]、SDA P12[5] および I2C1 – SCL P12[0]、SDA P12[1]。

I²C コンポーネント カスタマイザは**適切なピン割り当てを除き**、それらの条件を制御します。

動作方法

I²C ブロックはスリープ・モード時、I²C バスのトランザクションに応答します。I²C は、受信アドレスがスレーブ・アドレスと一致する場合、システムを起動します。アドレスが一致すると、ウェークアップ割り込みがアサートされてシステムを起動し、SCL が Low にプルダウンされます。ACK はシステム起動後に送信され、CPU はトランザクションの次の動作を決定します。

ウェークアップおよびクロック期間延長

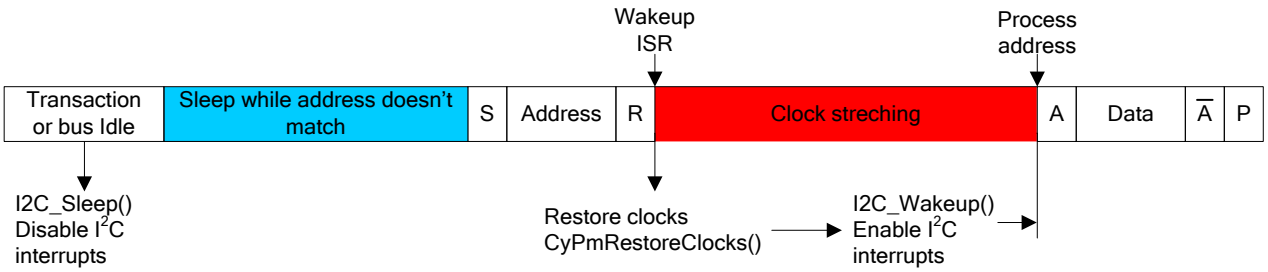
I²C スレーブはスリープ・モード終了時にクロック期間を延長します。システムのすべてのクロックは I²C トランザクションを継続する前に復元する必要があります。I²C 割り込みは、スリープに入る前はディスエーブルになり、I2C_Wakeup() 関数呼び出し後のみイネーブルになります。ウェークアップから I2C_Wakeup() の呼び出しの最後までの間は、SCL ラインが Low にプルダウンされます。

サンプル コード:

```
…
I2C_Sleep();            /* Go to Sleep and disable I2C interrupt */
CyPmSaveClocks();       /* Save clocks settings */

CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_I2C);

CyPmRestoreClocks();    /* Restore clocks */
I2C_Wakeup();           /* Wakeup, enable I2C interrupt and ACK the address, until
end of this call the SCL is pulled low */
…
```



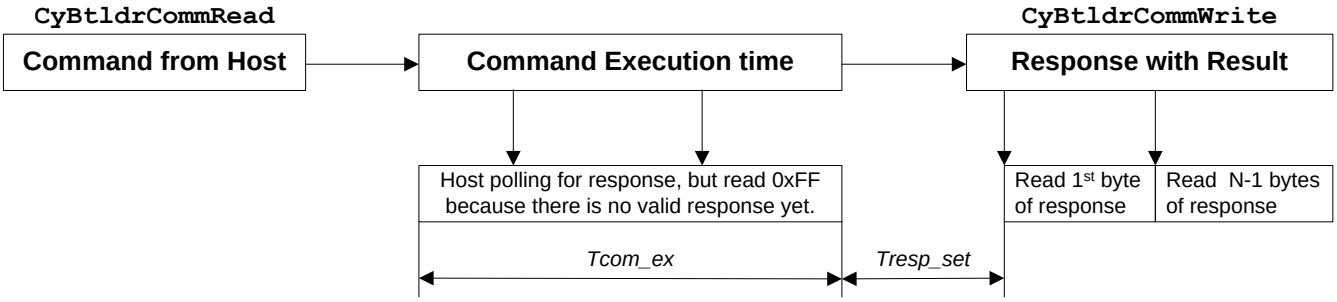
I²C 通信コンポーネントによるブートローダ・プロトコルインタラクション

ブートローダ・プロトコルはコマンド（書き込みトランザクション）および応答（読み取りトランザクション）として実装されます。

コマンドを発するホストと応答を送り戻すブートローダ間の時間がコマンド実行時間となります。ブートローダの I²C 通信コンポーネントはこのように設計されています：ホストが応答を必要とする場合、ブートローダはコマンドを実行し続け、0xFF を返します。

起動：I²C ブートローダ通信コンポーネントはコマンドの受信を求めており、有効な応答がまだありません。ホストからのすべての読み取りトランザクションは 0xFF を返します。すべての書き込みトランザクション以下のコマンドとして扱われます：

ブートローダ プロセス：ホストは 1 つの書き込みトランザクションでコマンドを発行し、応答のポーリングを開始します。I²C 通信コンポーネントは有効な応答がブートローダで渡されるまで 0xFF で答えます。0x01 を受信後、ホストは別の読み取りを実行して、残りの応答の N – 1 バイトを取得する必要があります。両方の読み取り完了後、結果は全応答パケットを結合して生成されます。

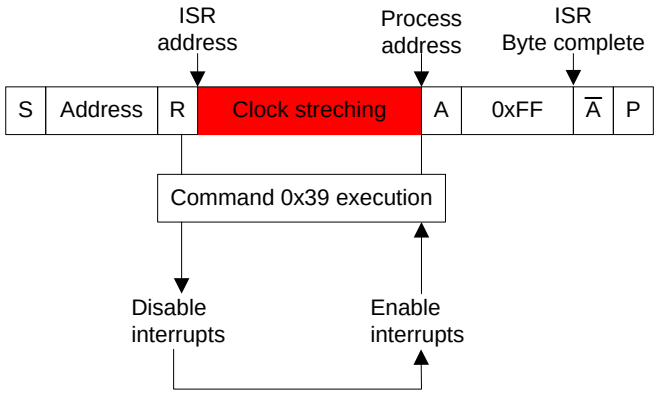


ホストは 1 バイトの読み取りでポーリングを実行する必要があります。さらにバイトを読み取ると応答を損なうことがあります。例えば、0xFF 0x01 0x03（1 バイトではなく、応答の 2 バイトが読み取られた）場合、全応答の次の読み取りは 2 つの無効なバイトを返します。なぜならこれらのバイトはすでに読み取られているからです（0x01 および 0x03）。

プーリングの回避方法: コマンド実行時間（Tcom_ex）と応答セットアップ時間（Tresp_set）をシステム設定（CPU 速度、コンパイラ、最適化レベル）に従って測定する必要があります。ホストはこの時間の後、応答を要

求する必要があります。コマンド実行時間はコマンド全体にわたって変わるため、より大きな時間を選択する必要があります。

プーリング中のクロック期間延長：I²C 通信コンポーネントは割り込みが動作中にイネーブルである必要があります。コマンド プログラム列（0x39）、これはフラッシュ データの 1 つの列をデバイスに書き込み、ディスエーブルにする割り込みを要求します。アドレスが I²C 通信コンポーネントによって許可されている場合、割り込みがディスエーブルである限り、クロックの期間延長が発生します。

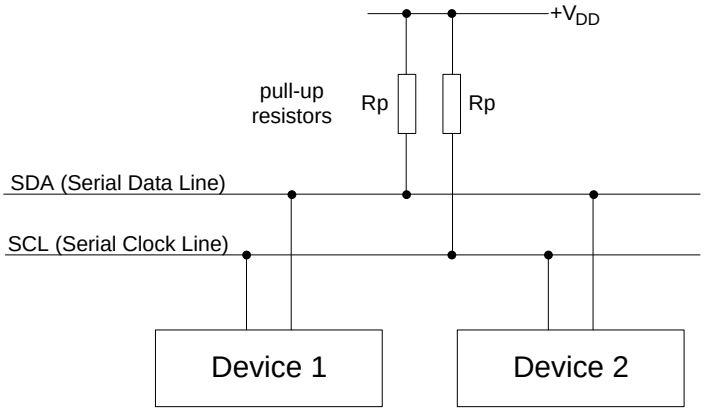


クロック期間延長の回避方法：クロック期間延長を回避するには、コマンド プログラム列（0x39）の実行時間（Tcom_ex）をシステム設定（CPU 速度、コンパイラ、最適化レベル）に従って測定します。ホストはこの時間の後、応答を要求する必要があります。

外部電気接続

図 3 で示すように、I²C バスには外部プルアップ抵抗が必要です。プルアップ抵抗(R_P)は、電源電圧、クロック速度、およびバスの容量に応じて決定されます。特定のデバイス（マスターまたはスレーブ）の最小の sink 電流は出力電圧の場合、V_{OLmax} = 0.4V で 3 mA 以上が必要です。これは 5 V システムの最小プルアップ抵抗値を約 1.5 kΩ に制限します。R_P の最大値は、バス上のキャパシタンスとクロック速度によって異なります。150 pF バスの容量を持つ 5 V システムの場合、プル アップ抵抗は 6 kΩ 以内です。プル アップ抵抗およびその他の物理バス仕様のサイズ変更の詳細については、www.nxp.com の NXP ウェブ サイトで『*I²C バス仕様*』を参照してください。

図 3. I²C バスへのデバイスの接続



注 I²C コンポーネントをサイプレスまたはサブライセンスを持つ関連業者から購入すると、Philips I²C の特許権の下でライセンスが付与されます。このライセンスにより、システムが Philips の定義する I²C の標準仕様に従う限り、I²C システムでこれらのコンポーネントを使用できます。2006 年 10 月 1 日以降、Philips Semiconductors 社は新社名 NXP Semiconductors を使用しています。

割り込みサービスルーチン

割り込みサービスルーチンはコンポーネントコードによって使用されます。変更しないでください。

次のユーザ・セクションでは以下のスレーブ処理が提供されます:

- カスタム内容および定義
- 追加アドレスの比較
- 事前読み取りバッファ

マスタ処理に提供されるユーザ・セクションはありません。

I²C コンポーネントはほとんど処理で割り込みを使用します。トランザクションのステータスはここで更新されます。ステータスの読み取りおよびクリアの関数は割り込みから保護されていません。
これらの関数を以下に記載します:

Master or multi-master:

- I2C_MasterStatus()
- I2C_MasterClearStatus()
- I2C_MasterGetReadBufSize()
- I2C_MasterGetWriteBufSize()

- I2C_MasterClearReadBuf()
- I2C_MasterClearWriteBuf()

Slave:

- I2C_SlaveStatus()
- I2C_SlaveClearReadStatus()
- I2C_SlaveClearWriteStatus()
- I2C_SlaveInitReadBuf()
- I2C_SlaveInitWriteBuf()
- I2C_SlaveGetReadBufSize()
- I2C_SlaveGetWriteBufSize()
- I2C_SlaveClearReadBuf()
- I2C_SlaveClearWriteBuf()

レジスタ

提供された関数は大多数のアプリケーションに必要な一般的なランタイム関数をサポートします。次のレジスタ・リファレンスには上級者向けの簡単な説明が提供されています。I2C_Data レジスタは API を使用せず、データを直接書き込むために使用される場合があります。これは CPU または DMA のいずれかを利用する際に便利なことがあります。

I2C コンポーネントの各構成で利用可能なレジスタは、固定ファンクションまたは UDB としての実装に従ってグループ分けされています。

固定ファンクションマスタ/スレーブ レジスタ

それらのレジスタの詳細については、『チップ・テクニカル・リファレンスマニュアル（TRM）』を参照してください。PSoC 3 製品版チップに追加されたすべてのビットは、以下の定義にアスタリスク（*）付きで示されています。

I2C_XCFG

拡張設定レジスタは、固定ファンクションハードウェア・ブロックでハードウェア・アドレス・モードおよびクロック・ソースの設定に利用できます。

ビット	7	6	5	4	3	2	1	0
値	csr_clk_en	i2c_on*	ready_to_sleep*	force_nak*	RSVD			hw_addr_en

- csr_clk_en: 固定ファンクションブロックのコア論理のゲート制御をイネーブルするのに使用されます。
- i2c_on*: I²C ブロックのウェークアップ ソースとしての選択に使用されます。
- ready_to_sleep*: ブロックのスリープ準備の通知に使用されます。
- force_nak*: トランザクションを強制的に NAK するのに使用されます。
- hw_addr_en: ハードウェア・アドレス比較モードをイネーブルするのに使用されます。

I2C_ADDR

スレーブ・アドレス・レジスタは、XCFG レジスタがイネーブルの場合、固定関数ハードウェア・ブロックでハードウェア比較モードのスレーブ・デバイス・アドレスの設定に使用できます。

ビット	7	6	5	4	3	2	1	0
値	RSVD	slave_address						

- slave_address: ハードウェア・アドレス比較モードの 7 ビット スレーブ・アドレスの定義に使用されます。

I2C_CFG

構成レジスタは、固定ファンクションハードウェア・ブロックでの基本機能の設定に使用できます。

ビット	7	6	5	4	3	2	1	0
値	sio_select	pselect	bus_error_ie	stop_ie	clock_rate[1:0]		en_mstr	en_slave

- sio_select: SCL および SDA の SIO1 ラインか SIO2 のラインから選択するために使用されます。pselect はこのビットに効果があるように設定する必要があります。
- pselect: SIO 直接接続、または SCL と SDA ラインの GPIO/SIO ピンに接続された DSI の選択に使用されます。
- bus_error_ie: bus_error の割り込み生成をイネーブルするのに使用されます。
- stop_ie: ストップ・ビット検出の割り込み生成をイネーブルするのに使用されます。
- clock_rate: オーバーサンプリングの 16 ビットまたは 32 ビットの選択に使用されます。PSoC 3 製品版はビット 2 のみです。
- en_mstr: マスタ・モードのイネーブルに使用されます。
- en_slave: スレーブ・モードのイネーブルに使用されます。

I2C_CSR

コントロールおよびステータスレジスタは、固定ファンクションハードウェア・ブロックでランタイム制御および状態フィードバックに使用できます。

ビット	7	6	5	4	3	2	1	0
値	bus_error	lost_arb*	stop_status	ack	アドレス	送信	lrb	byte_complete

- bus_error: バス エラー検出状態ビット。これはこのビット位置に「0」を書き込むことでクリアする必要があります。
- lost_arb*: アービトレーション・ロスト検出状態ビット。
- stop_status: Stop 検出状態ビット。これはこの位置に「0」を書き込むことでクリアする必要があります。
- ack: 応答コントロール・ビット。このビットは受信した最後のバイトを ACK にするには 1 に設定、または受信した最後のバイトを NAK にするには 0 に設定する必要があります。
- address: 受信したバイトがアドレス・バイトであった場合に設定します。
- transmit: ファームウェアを使用してバイトの転送方向を定義します。
- lrb: 最後に受信したビット状態。このビットは送信された最終バイトのレシーバから返された 9 番目のビット（ACK/NAK）の状態を示します。
- byte_complete: このレジスタの最後のリード後の送受信ステータス。送信モードで、このビットは、最後に読み取った後に送信された 8 ビットのデータと ACK/NAK を表示します。受信モードで、このビットは、このレジスタを最後に読み取った後に受信された 8 ビットのデータを表示します。

I2C_DATA

データ レジスタは、固定ファンクションハードウェア・ブロックでデータのランタイム送信および受信に使用できます。

ビット	7	6	5	4	3	2	1	0
値	データ							

- data: 送信モードで、このレジスタには送信するデータが書き込まれます。受信モードでは、byte_complete のステータスを受信すると、このレジスタが読み込まれます。

I2C_MCSR

マスタコントロールおよびステータスレジスタは、固定ファンクションハードウェア・ブロックでマスタ・モード処理のランタイム制御およびステータス フィードバックのために用意されています。

ビット	7	6	5	4	3	2	1	0
値	RSVD			stop_gen*	bus_busy	master_mode	restart_gen	start_gen

- stop_gen*: セットされている場合、マスタモードで、最後のバイト転送の時に Stop が生成されます。
- bus_busy: バスの状態を示します。0 は Stop コンディションが検出されたことを意味し、1 は Start コンディションが検出されたことを示しています。
- master_mode: 有効な Start コンディションが生成されて、ハードウェア・デバイスがバス マスタとして動作していることを示します。
- restart_gen: バス上に Restart コンディションを生成するためのコントロールレジスタ。このビットは Restart が実行された後でハードウェアによってクリアされます（状態の完了をポーリングとして設定した後にステータスとして読むことができます）。
- start_gen: レジスタを制御してバス上に Start 条件を生成します。このビットは Start が実行された後でハードウェアによってクリアされます（状態の完了をポーリングとして設定した後、ステータスとして読み取ることができます）。

UDB マスタ

UDB レジスタ定義は I²C の Verilog 実装から派生します。これらのレジスタの定義について詳しくは、特定モードの Verilog 実装を参照してください。

I2C_CFG

コントロールレジスタはハードウェアのランタイム制御の UDB 実装で使用できます。

ビット	7	6	5	4	3	2	1	0
値	start_gen	stop_gen	restart_gen	ack	RSVD	送信	en_master	RSVD

- start_gen: 1 をセットするとバス上に Start コンディションを生成します。このビットは、次のトランザクションの起動前にファームウェアでクリアする必要があります。
- stop_gen: 1 を設定するとバス上に Stop コンディションを生成します。このビットは、次のトランザクションの起動前にファームウェアでクリアする必要があります。
- restart_gen: 1 をセットするとバス上に Restart コンディションを生成します。このビットは、Restart 条件の生成後にファームウェアでクリアする必要があります。
- ack: 1 をセットすると次の読み取りバイトに NAK 応答します。クリアすると次の読み取りバイトに ACK 応答します。このビットは、バイト間をファームウェアでクリアする必要があります。

- transmit: 1 をセットすると現在のモードを送信に設定し、0 にクリアすると、データのバイトを受信します。このビットは、次のトランザクションの送信または受信前にファームウェアでクリアする必要があります。
- en_master: 1 に設定してマスタの機能をイネーブルにします。

I2C_CSR

ステータスレジスタは、ハードウェアからランタイム・ステータスのフィードバック用として UDB 実装で使用できます。ステータス・データは、Mode = 1 とされたすべてのビットは、カウンタの入力クロック エッジで取り込まれます。これらのビットはスティッキーで、ステータスレジスタの読み取り時にクリアされます。Mode = 0 と設定されている全ビットは入力からステータスレジスタへ直接読み取られます。これらはスティッキーではないため、読み取り時にクリアされません。Mode = 1 と設定されているビットには、下記の定義でアスタリスク(*)が付されています。

ビット	7	6	5	4	3	2	1	0
値	RSVD	lost_arb*	stop_status*	bus_busy	アドレス	master_mode	lrb	byte_complete

- lost_arb*: セットされている場合、アービトレーションがロストしたことを示しています(マルチマスタおよびマルチマスタスレーブ・モード)。
- stop_status*: 設定されている場合、Stop 条件がバス上で検出されたことを示しています。
- bus_busy: セットされている場合、バスがビジーであることを示しています。データは現在、送信または受信中です。
- address: アドレス検出 セットされている場合、アドレス・バイトが送信されたことを示しています。
- master_mode: 有効な Start コンディションが生成されて、ハードウェア・デバイスがバス マスタとして動作していることを示します。
- lbr: Last Received Bit (最後に受信したビット): 最後に受信したビットの状態を示します。これは最後に送信したバイトに対して受信された ACK/NAK です。クリア = ACK およびセット = NAK.。
- byte_complete: この最後のレジスタを読み込んでから、ステータスを送信または受信します。送信モードで、このビットは 8 ビットのデータと ACK/NAK が、最後の読み取りの後で送信されたことを示しています。受信モードで、このビットは 8 ビットのデータが、このレジスタの最後の読み取りの後で受信されたことを示しています。

I2C_INT_MASK

割り込みマスク レジスタは、UDB 実装において、どちらのステータスビットが割込みソースとしてイネーブルするのかを指定するのに使用できます。いずれのステータスレジスタ・ビットも、I2C_CSR でステータスレジスタのビットフィールド定義との一対一の関係を持つ割り込みソースとしてイネーブルにすることができます。

I2C_ADDRESS

スレーブ・アドレス・レジスタは、UDB 実装で使用でき、ハードウェア比較モードのスレーブ・デバイス・アドレスを設定できます。

ビット	7	6	5	4	3	2	1	0
値	RSVD	slave_address						

- slave_address: ハードウェア比較モードの 7 ビットスレーブ・アドレスの定義に使用されます

I2C_DATA

データ レジスタは、ランタイム送信およびデータ受信の UDB 実装ブロックで使用でききます。

ビット	7	6	5	4	3	2	1	0
値	データ							

- data: 送信モードで、このレジスタには送信するデータが書き込まれます。受信モードでは、byte_complete のステータスを受信すると、このレジスタが読み込まれます。

I2C_GO

Go レジスタは、マスタが送信する場合、データ レジスタのデータを強制送信します。Go レジスタは、マスタが受信する場合、データ レジスタのデータを強制受信します。このレジスタへの書き込みは、どんな値であっても、すべてこの動作を強制します。

UDB スレーブ

UDB レジスタ定義は I²C の Verilog 実装から派生します。これらのレジスタの定義について詳しくは、特定モードの Verilog 実装を参照してください。

I2C_CFG

コントロールレジスタはハードウェアのランタイム制御用として UDB 実装で使用できます。

ビット	7	6	5	4	3	2	1	0
値	RSVD	RSVD	RSVD	nak	any_address	送信	RSVD	en_slave

- nak: セットされた場合、受信した最後のバイトに NAK 応答させるのに使用されます。このビットは、各バイト間でファームウェアでクリアする必要があります。
- any_address: セットされた場合、デバイスをイネーブルにして、I2C_ADDRESS で提供された単一アドレスではなく、受信するどのデバイス・アドレスにも応答します。

- transmit: データを送信または受信するモードの設定に使用されます。このビットは、各バイト間でファームウェアでクリアする必要があります。設定 = 送信およびクリア済み = 受信。
- en_slave: 1 にセットするとスレーブの機能をイネーブルにします。

I2C_CSR

ステータスレジスタは、ハードウェアからランタイム・ステータスのフィードバック用として UDB 実装で使用できます。ステータス・データは、Mode = 1 で設定されたすべてのビットについてカウンタの入力クロック エッジで取り込まれます。これらのビットはスティッキーで、ステータスレジスタの読み取り時にクリアされます。Mode = 0 として設定されている他のすべてのビットは入力からステータスレジスタへ直接読み取られます。これらはスティッキーではないため、読み取り時にクリアされません。Mode = 1 として設定されたすべてのビットは、以下の定義にアスタリスク(*)付きで示されています。

ビット	7	6	5	4	3	2	1	0
値	RSVD	RSVD	stop*	RSVD	アドレス	RSVD	lrb	byte_complete

- stop*: セットされている場合、Stop コンディションがバス上で検出されたことを示しています。
- address: アドレス検出 セットされた場合、アドレス・バイトが受信されたことを示します。
- lbr: Last Received Bit (最後に受信したビット): 最後に受信したビットの状態を示します。これは最後に送信されたバイトに対して受信された ACK/NAK です。クリア = ACK およびセット = NAK.
- byte_complete: 最後にこのレジスタをリードした後の送受信ステータス。送信モードで、このビットは 最後のリードの後に贈られた 8 ビットのデータと ACK/NAK を示します。受信モードで、このビットは 8 ビットのデータが、このレジスタの最後の読み取りの後で受信されたことを示しています。

I2C_INT_MASK

割り込みマスク レジスタは、UDB 実装で使用でき、どのステータスビットを割込みソースとしてイネーブルにするか指定できます。いずれのステータスレジスタ・ビットも、I2C_CSR レジスタでステータスレジスタのビットフィールド定義との一対一の関係を持つ割り込みソースとしてイネーブルにすることができます。2 つの割り込みソースが動作中に使用されます: byte_complete および Stop。

I2C_ADDRESS

スレーブ・アドレス・レジスタは、UDB 実装で使用でき、ハードウェア比較モードのスレーブ・デバイス・アドレスを設定できます。

ビット	7	6	5	4	3	2	1	0
値	RSVD	slave_address						

- slave_address: ハードウェア比較モードの 7 ビットスレーブ・アドレスの定義に使用されます

I2C_DATA

データ レジスタは、ランタイム送信およびデータ受信の UDB 実装ブロックで使用でききます。

ビット	7	6	5	4	3	2	1	0
値	データ							

- data: 送信モードで、このレジスタには送信するデータが書き込まれます。受信モードでは、byte_complete のステータスを受信すると、このレジスタが読み込まれます。

I2C_GO

Go レジスタは、マスタが送信する場合、データレジスタのデータを強制送信します。Go レジスタは、マスタが受信する場合、データレジスタにデータ を強制受信します。このレジスタへの書き込みは、どんな値であっても、すべてこの動作を強制します。

DC 電気的特性と AC 電気的特性（FF 実装）

以下の値は、期待される性能を示しており、初期特性データを基にしています。

I²C の DC 仕様

パラメータ	説明	条件	最小値	Typ	最大値	単位
	ブロックの電流消費	イネーブル、100 kbps に構成	--	--	250	µA
		イネーブル、400 kbpsに構成	--	--	260	µA
		スリープモードからのウェイク	--	--	30	µA

I²C の AC 仕様

パラメータ	説明	条件	最小値	Typ	最大値	単位
	ビット レート		--	--	1	Mbps

DC 電気的特性と AC 電気的特性（UDB 実装）

以下の値は、期待される性能を示しており、初期特性データを基にしています。

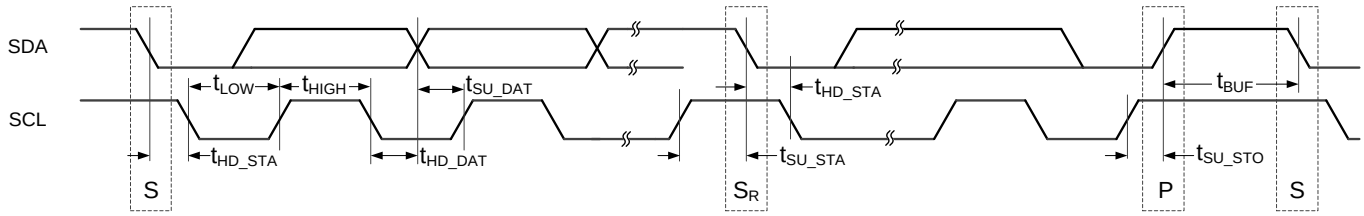
「すべてのルーティングでの最大」タイミング特性

パラメータ	説明	最小値	Typ	最大値	単位
f_{SCL}	SCL クロック周波数				

パラメータ	説明		最小値	Typ	最大値	単位
		標準モード	–	100	–	kHz
		Fastモード	–	400	–	kHz
		Fastモード プラス	–	1000	–	kHz
f_{CLOCK}	コンポーネント入力クロック周波数		–	16 × f_{SCL}	–	kHz
t_{LOW}	SCL クロックの LOW 期間		–	8	–	t_{CY_clock} [7]
t_{HIGH}	SCL クロックの HIGH 期間		–	8	–	t_{CY_clock} 7
t_{HD_STA}	ホールド時間（Repated） STARTコンディション		–	15	–	t_{CY_clock}
t_{SU_STA}	繰り返す STARTコンディションのセットアップ時間		–	9	–	t_{CY_clock}
t_{HD_DAT}	データ ホールド時間		–	1	–	t_{CY_clock}
t_{SU_DAT}	データ セットアップ時間		–	7	–	t_{CY_clock}
t_{SU_STO}	STOPコンディションのセットアップ時間		–	9	–	t_{CY_clock}
t_{BUF}	STOPコンディションと STARTコンディションとの間のバス空き時間		–	32	–	t_{CY_clock}
t_{RESET}	リセット パルス幅		–	2	–	t_{CY_clock}

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

図 4. データ トランザクションのタイミング図



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。最大値は、以下のメカニズムで、STA 結果を使って、それぞれの設計用に計算できます。

f_{CLOCK} 最大コンポーネント・クロック周波数は、クロックサマリ中の Timing results で、component clock（この場合は CLK)として提供されています。最大コンポーネント・クロックは $f_{CLOCK} = 16 \times f_{SCL} = 16 \times 1000$ kHz = 16 MHz に限定されるため、

STA レポートはレポート最大クロック周波数（Max Freq）が違反していないことの確認のみに使用してください。*_timing.html* ファイルからの、コンポーネント・クロックの制限例は、以下の通りです。

+Clock Summary

Clock	Actual Freq	Max Freq	Violation
BUS_CLK	48.000 MHz	112.664 MHz	
Clock	16.000 MHz	20.571 MHz	

残りのパラメータは、実装固有のものであり、クロックサイクル単位で表されます。I²C コンポーネントは I²C バス仕様の 2007 年からの改訂 3 と互換性があります。

t_{SCL} は最大 1000 kbps までの I²C データ転送速度値を定義します。標準データ転送速度は 50、100、400、および 1000 kbps です。16x 入力クロックは必要なデータ転送速度の取得を要求します。

t_{LOW} SCL クロックの LOW 期間。コンポーネントは 50 パーセントのデューティ・クロック・サイクルを生成します。

t_{HIGH} SCL クロックの HIGH 期間。コンポーネントは 50 パーセントのデューティ・クロック・サイクルを生成します。

t_{HD_STA} SDA の HIGH から LOW への移行後に Start 条件を生成するのに SCL 信号が HIGH となる最小時間。この時間が経過した後、最初のクロック・パルスが生成されます。

t_{SU_STA} SDA の HIGH から LOW への移行前に Start 条件を生成するのに SCL 信号が HIGH となる最小時間。

t_{HD_DAT} SCL 信号の立ち下がりエッジ後に、データが有効になる最小時間。

t_{SU_DAT} SCL 信号の立ち下がりエッジ前に、データが有効になる最小時間。

t_{SU_STO} SDA 信号の LOW から HIGH への移行前に Stop 条件を生成するのに SCL 信号が HIGH となる必要な最小時間。

t_{BUF} バスが Stop 条件後にフリーと見なされる期間。

t_{RESET} コンポーネント実装は 2 サイクル幅のリセット信号を要求します。

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
3.1	I2C_SSTAT_RD_CMPT から I2C_SSTAT_RD_CMPLT までの定義を変更 I2C_SSTAT_WR_CMPT から I2C_SSTAT_WR_CMPLT までの定義を変更	読み取りおよび書き込み完了フラグのマスタ定義を準拠するには。コンポーネントは両方の定義をサポートしますが、I2C_SSTAT_RD_CMPT および I2C_SSTAT_WR_CMPT は廃止されます。
	.cyre ファイルのすべての API に CYREENTRANT キーワードを追加しました。	すべての API が真に再入可能とは限りません。コンポーネント API ソースファイルに含まれているコメントは、どの関数が候補でないかを示しています。 この変更には、安全な（フラグまたはクリティカル セクションによる並行コールから保護された）方法で使用されるリエントラントではない関数のコンパイラの警告を制限する必要があります。
3.0.a	データシートのマイナーな編集と更新	
3.0	カスタマイザの外観を変更	より直観的で使いやすくなりました。
	UDB クロックトレランスを追加しました。	様々な設定に対するクロック警告の出現を回避。
	Sleep オプションから Enable にされた FF 移植のコンポーネントは、ハイバネート終了後に正しく設定を復元します。	ハイバネート・モードのコンポーネントの振る舞いを修正します。
	I²C 割り込みは I2C_Start() が呼び出された後でイネーブルになります。	ユーザがスレーブ・モードの I2C_Start() 後の割り込みをイネーブルにするのを忘れた場合、エラーは表示されません。
	UDB 実装の内部クロックのサポートを追加しました。	機能拡張。
	I2C_SlaveGetWriteByte() および I2C_SlavePutReadByte() 関数を削除しました。	これらの関数は使用できません。
2.20	コンポーネントの UDB ベース実装にブートローダ通信サポートを追加しました。	デザインのブートロードをサポートする複数の I²C コンポーネントが利用可能です。これはcy_boot v2.21 に含まれるカスタム・ブートローダ機能により使用できます。
	ゼロ データホールドタイムによって、トランザクション中のStart 条件誤検出の修正。	スレーブはマスタからのゼロ データホールドタイムで正常に動作します。
2.10	マルチマスタスレーブ・モードを追加しました。	マルチマスタスレーブ機能のサポートがコンポーネントに追加されます。
	カスタマイザのラベルおよび説明の編集	フィーリングおよびコンポーネント カスタマイザの内容を改善します。

バージョン	変更の説明	変更の理由 / 影響
	I²C ブートローダ通信コンポーネントの振る舞いを変更し、読み取り時にクロック期間延長を抑制します。	I²C ブートローダ通信コンポーネントは起動プロセス前に読み取りコマンドが表示される場合は常に SCL low を保持します。
	データシートに特性データを追加しました。	
	データシートのマイナーな編集と更新	
2.0.a	コンポーネントをcomponent catalogのサブフォルダに移動しました。	
	データシートのマイナーな編集と更新	
2.0	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。	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イネーブル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。
	PSoC 3 製品版および上記をサポートするため、コンポーネントを更新しました。[Configure] ダイアログが以下のように更新されました:	PSoC 3 製品版デバイスをサポートする新規要件から、この新しいバージョン 2.0 が作成されました。 バージョン 1.xx は PSoC 3 ES2 および PSoC 5 シリコンリビジョンをサポートします。
	I²C アドレス一致機能をウェークアップするための I2C ピン接続ポート設定を追加しました。	I²C コンポーネントは I²C アドレス一致により、スリープ・モードからデバイスを起動できます。
	データベースを更新しました。	パラメータとセットアップ、クロック選択、およびリソース選択をUDB 実装に反映するため更新しました。 サンプル コードのエラーが修正されました。
	リエントラントのサポートをコンポーネントに追加します。	必要な場合に、特定の API を再入可能できるようにするためです。

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。